Kalle Karonmaa

Performance evaluation of software switching using commodity hardware

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 19.04.2012

Thesis supervisor:

Prof. Jukka Manner

Thesis instructor:

M.Sc. (Tech.) Nuutti Varis

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Kalle Karonmaa

Title: Performance evaluation of software switching using commodity hardware

Date: 19.04.2012 Language: English Number of pages:11+72

Department of Communications and Networking

Professorship: Networking technology Code: S-38

Supervisor: Prof. Jukka Manner

Instructor: M.Sc. (Tech.) Nuutti Varis

Software based packet forwarding using commodity hardware has huge potential to become a viable option in smaller networks. Compared to hardware based on ASICs and network processors, the software option has significant advantages in both flexibility and price. The downside of software routers is obviously their performance compared to traditional hardware options. The field is also still quite unknown within the group of networking specialists, albeit most routers and WLAN boxes found at homes forward packets using software.

The goal of this thesis is to evaluate the performance of commodity hardware under a switching load using the Linux bridging implementation. In addition, different ways of constructing a router or switch are explored and the Intel Nehalem microarchitecture presented. The relevant parts of the Linux network stack and optimization of both hardware and software performance are also covered. The actual performance evaluation is done using two servers: one with a single processor and another with dual processors.

Optimizing performance by changing the Linux kernel configuration, BIOS settings and driver parameters led to the single processor server forwarding traffic at a maximum rate of 7.2 million frames per second compared to 10 million for the dual processor server. Both these values are still far from the theoretical maximums, which lead to the search for bottlenecks. The main bottlenecks were found out to be the processor clock frequency and memory channel width. They both clearly limited the performance.

Keywords: Software routing, software switching, performance, Linux

AALTO-YLIOPISTO
SÄHKÖTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Kalle Karonmaa

Työn nimi: Ohjelmistopohjaisen kytkimen suorituskyvyn arvionti

Päivämäärä: 19.04.2012 Kieli: Englanti Sivumäärä:11+72

Tietoliikenne- ja tietoverkkotekniikan laitos

Professuuri: Tietoverkkotekniikka Koodi: S-38

Valvoja: Prof. Jukka Manner

Ohjaaja: FM Nuutti Varis

Ohjelmistopohjaisella pakettien välityksellä on suuri potentiaali syrjäyttää tyypillisesti ASIC:lla ja verkkoprosessoreilla toteutetut laitteet. Sen suurimmat edut näihin kilpaileviin tekniikoihin verrattuna ovat helppo ohjelmoitavuus ja edullinen hinta. Huonona puolena voidaan pitää suhteellisen heikkoa suorituskykyä. Myös tietoverkko-osaajien joukossa pakettien välittäminen ohjelmistopohjaisesti on vielä suhteellisen tuntematon käsite, vaikkakin suurin osa kotoa löytyvistä reitittimistä ja WLAN tukiasemista välittävät paketteja juuri tätä tapaa hyödyntäen.

Tämän diplomityön tavoitteena on mitata Linuxin suorituskykyä kytkimenä, käyttäen kaupallisesti saatavilla olevaa laitteistoa. Tämän lisäksi työssä käsitellään eri tapoja reitittimen tai kytkimen rakentamiseksi ja esitellään Intelin Nehalem mikroarkkitehtuuri. Myös olennaisimmat osat Linuxin verkkopinosta käydään läpi, minkä lisäksi käsitellään sekä laitteiston että ohjelmiston optimointia suorituskyvyn kannalta. Varsinaiset suorituskykymittaukset tehdään kahta eri palvelinta käyttäen: toinen palvelimista käyttää yhtä suoritinta toisen toimiessa kahden suorittimen varassa.

Suorituskykyä optimoitiin muuttamalla Linuxin ytimen konfiguraatiota ja sekä BIOS:n että ajurien asetuksia muokkaamalla. Näitä muutoksia tekemällä saatiin yhden suorittimen palvelimesta kulkemaan läpi parhaimmillaan 7.2 miljoonaa kehystä sekunnissa, kun taas kahden suorittimen palvelin kykeni 10 miljoonaan kehykseen sekunnissa. Molemmat luvut jäävät kauas teoreettisesta maksimiläpäiskyvystä, minkä takia palvelimista yritettiin löytää pullonkaulat, jotka rajoittavat suorituskykyä. Pullonkauloiksi todettiin suorittimen kellotaajuus sekä muistikanavien lukumäärä.

Avainsanat: Ohjelmistopohjainen reititys, suorituskyky, Linux

Preface

The research for this thesis was conducted at the Department of Communications and Networking at the Aalto University, School of Electrical Engineering.

I would like to thank my supervisor, professor Jukka Manner, for the feedback on my work and for making it possible to do research on this matter in the first place. I would also like to thank my instructor Nuutti Varis for helping out with all the practicalities related to my research and for the patience in answering all my questions.

I would like to express my deepest gratitude to my parents for all their support and encouragement throughout my years of studying. To all of my friends; thank you for the laughter and good times that have kept my thoughts away from the writing process. Last but certainly not least, Satu, thank you for being there for me through both good times and bad. I could not have made it without you.

Espoo, 19.4.2012

Kalle Karonmaa

Contents

Acronyms

ACPI . . . . . Advanced Configuration and Power Interface

AMD . . . . . Advanced Micro Devices

ASIC . . . . . Application-Specific Integrated Circuit

ASIP . . . . . Application Specific Instruction Set Processor

ASSP . . . . . Application Specific Standard Product

AVX . . . . . Advanced Vector Extensions

BIOS . . . . . Basic Input/Output System

BSD . . . . . Berkeley Software Distribution

COTS . . . . . Commodity off The Shelf

CPU . . . . . Central Processing Unit

DDR3 . . . . . Double Data Rate type three

DRAM . . . . . Dynamic Random Access Memory

DUT . . . . . Device Under Test

EIST . . . . . Enhanced Intel SpeedStep Technology

FPGA . . . . . Field Programmable Gate Array

FSB . . . . . Front Side Bus

GB/s . . . . . Gigabytes per second

GPU . . . . . Graphics Processing Unit

GRO . . . . . Generic Receive Offload

GT/s . . . . . Gigatransfers per second

HTTP . . . . . Hypertext Transfer Protocol

I/O . . . . . Input/Output

I/OAT . . . . . I/O Acceleration Technology

IETF . . . . . Internet Engineering Task Force

IP . . . . . Internet Protocol

IRQ . . . . . Interrupt Request

ISA . . . . . Instruction Set Architecture

ITR . . . . . Interrupt Throttle Rate

LRO . . . . . Large Receive Offload

MAC . . . . . Media Access Control

MMX . . . . . Multi Media Extensions

MTU . . . . . Maximum Transmission Unit

NAPI . . . . . Linux New API

NIC . . . . . Network Interface Card

NoC . . . . . Network on Chip

NPF . . . . . Network Processing Forum

NPU . . . . . Network Processing Unit

NUMA . . . . . Non Uniform Memory Access

OIF . . . . . Optical Internetworking Forum

OS . . . . . Operating System

PC . . . . . Personal Computer

PCI . . . . . Peripheral Component Interconnect

PCI-SIG . . . . . PCI Special Interest Group

PCIe . . . . . PCI Express

PU . . . . . Processing Unit

QoS . . . . . Quality of Service

QPI . . . . . QuickPath Interconnect

RFC . . . . . Request For Comments

RSS . . . . . Receive-Side Scaling

SDRAM . . . . . Synchronous DRAM

SFD . . . . . Start Frame Delimiter

SFI . . . . . Simple Firmware Interface

SRAM . . . . . Static Random Access Memory

SSE . . . . . Streaming SIMD

STP . . . . . Spanning Tree Protocol

TCP . . . . . Transport Control Protocol

URL . . . . . Uniform Resource Locator

WLAN . . . . . Wireless Local Area Network

List of Figures

List of Tables

1 Introduction

Packet forwarding, whether it be on layer 2 or 3, could be called the cornerstone of the Internet. Without the link-layer and network protocols, routers and, switches one wouldn't be able to browse any web pages or participate in any of the other activities that nowadays have become so obvious and easy to use, that one rarely ponders what really is going on behind the scenes to make this experience possible. Traditionally routers and switches have been manufactured by giants like Cisco [1] and Juniper [2]. These expensive machines with terabits of throughput capacity, purpose made hardware and price tags ranging up to hundreds of thousands of euros have kept the Internet running from its beginning.

The downsides of traditional hardware routers and switches are many. The power consumption is enormous, the devices offer only the services that the manufacturer has chosen and as already stated the prices are relatively high. Obviously network operators and other organizations that require line rate throughput and five nines of uptime for hundreds of connections will still be using these machines for quite a while. But what about smaller organizations with less stringent requirements? This is the market where a software router or switch would be most suitable.

The advantages of a software router over a regular one are many. A software router operates on Commodity off The Shelf (COTS) Personal Computer (PC) hardware that is available from several manufacturers at reasonable prices. Most software routers operate using free open source software, which enables the customers to make modifications to the software according to their own needs. This is not of course necessary as the available router software offer most basic routing features and beyond by default. The open source nature of the software is probably at the same time the biggest limiting factor to the success of software routers. Using free open source software it is very hard to get any real support for your product, should a situation requiring it arise.

A company called Vyatta [3] offers their customers software based routers built upon regular PC hardware that use custom made open source software. These products offer layer 3 forwarding throughput of some 250000 to 3000000 packets per second, which is certainly enough for some smaller applications. Software routers are still more common than one would think. Most small routers, switches and Wireless Local Area Network (WLAN) boxes used at homes run on a general purpose processor and some optimized Linux based software. Software routers have traditionally also been used in the academic world to run some of the school networks. An example of this is Uppsala university that has used various open source routers in production since 1999 [4].

A problem of the software routers using Linux is its network stack. At the time

of its creation the stack was not optimized for routers and switches that required large amounts of throughput. This has lead to a vast amount of research on the area of optimizing the network stack. One interesting result of such research is PacketShader [5] that uses a regular graphics card in addition to an optimized network stack to accelerate the processing of packets. Another example of a software router architecture that has chosen to implement a new network stack is the Click modular router [6]. Click is also used as a basis for many of the software router implementations, as its modular nature makes it easy to implement new functions for it. One interesting software router implementation using Click is RouteBricks [7], that by clustering several servers to act as one router, has gained some performance improvements.

All the mentioned software router projects are quite complicated and require a lot of expertise and experience to get them running. That is why this thesis concentrates on the performance evaluation of the software that can be found in any Linux distribution namely the built in layer 2 packet switching implementation (Linux Bridge). Thus the aim of this thesis is to measure the throghput performance of the Linux operating system, working as a switch, using commodity x86 hardware. Another goal is to try and modify settings regarding the different aspects of the kernel, BIOS and network adapter device driver to get the best possible performance out of the system. Finally the results will be analyzed and some reasoning as to why the results look like they do will be presented.

The measured results show that a remarkable increase in performance can be gained by doing some rather simple changes to the configuration of the equipment. By doing a certain set of changes, the throughput of 64 byte frames increased from some 3 million frames per second to around 7 million frames per second for the single processor server. The dual processor server showed an increase around 4 to 10 million frames per second with the same frame size. Using 1518 byte frames, both servers reached line rate throughput. A deeper look at the results will be given in Chapter 4.

This thesis consists of five chapters and two appendices. Chapter 1 introduces the reader to the subject of software routers and presents the research questions that will be answered. Chapter 2 gives some more insight to the different ways of constructing a router and, a deeper look at the x86 architecture and optimizing it. Chapter 3 explains the journey of a packet through the Linux network stack, some of the key technologies created to increase processing performance and takes a look at optimizing performance of the Operating System (OS). Chapter 4 introduces the test setup and results, and finishes by analyzing the results. Chapter 5 concludes this thesis while Appendices A and B include the exact numerical results for the measurements.

2 Network nodes and hardware

This chapter gives more background to the different technologies used to build routers or switches, namely Application-Specific Integrated Circuits (ASICs), Network Processing Units (NPUs) and commodity PC or x86 hardware. All of them have their benefits and downsides, some of which will be presented in the following sections. The Intel Nehalem x86 architecture will also be presented in detail, as servers built upon it are used for the tests that form the basis of this thesis. First, a simplified model of how a router works is presented.

2.1 Inside routers and switches

The aim of this section is to give an overview on what happens to a packet inside a router or switch, [1] while the other sections aim to present different building blocks that networking equipment can be constructed of. This section presents a typical router design but the same principle can also be applied to switches. In general a router can be presented using four different main components; the input ports, switch fabric, routing processor and output ports [8]. Figure 1 shows the interactions of these components.

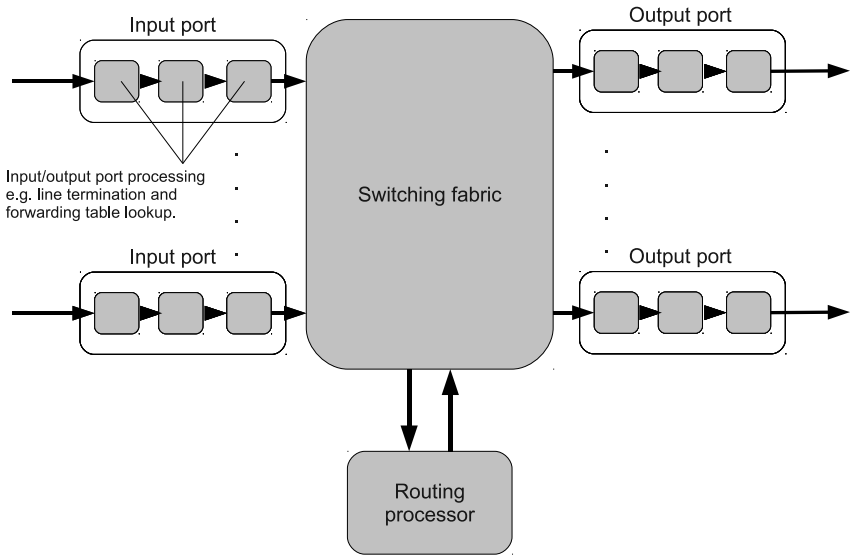

Figure 1: A simplified block diagram of a router or switch.

When a packet first enters the router it comes to an input port. Typically each input port is physically paired with an output port, which enables full duplex traffic. For simplicity these are treated separately in this text and picture. The input port

[1] Switches are also commonly called bridges.

is the part that terminates the physical connection on the input link and handles the data link layer processing. It also typically houses a copy of the forwarding table that is used to make the decision to which output port the incoming packet is to be forwarded to. Letting each input port do its own forwarding decision removes a potential bottleneck inside the router compared to the case where a single entity would be responsible for the forwarding decisions. The mentioned technique is most commonly found in high-end routers, while lower grade equipment and software routers rely on a centralized approach, where one entity makes all the forwarding decisions.

When the forwarding decision for the packet entering a router has been made, it can be moved from the input port to the switching fabric. The switching fabric does the actual transfer of the packet from an input to an output port. It can be designed using a few different methods depending on the performance requirements. The first method is switching via memory where the input port copies the packet to a certain memory location that the output port then reads. This is the technique used by software routers. Another approach is to use a shared bus that connects all of the ports on the router. In this approach one packet at a time is moved from an input port to an output port, which makes the bus the limiting factor of the throughput performance of the router. The final approach is to build an interconnection network of buses, which in its simplest form might be for example a crossbar. The network enables packets to be forwarded to several output ports at the same time. If the switching fabric cannot handle the situation where several input ports are forwarding packets to one output port at the time, queuing needs to be implemented at the input ports.

Finally after having passed the switching fabric the packet arrives at the output port. The output port performs the same functions as an input port but obviously in the reverse direction. In addition it needs to perform queuing of packets in the case that several input ports send a packet to one output port at the same time. This is also where packet processing related to Quality of Service (QoS) and scheduling is done. All the functions related to packet switching are typically referred to as data path functions [9]. The routing processor handles all the control related functions of the router such as configuration, running routing protocols and executing forwarding table updates. These are commonly referred to as control path functions.

Technically speaking routers and switches work in a similar fashion. The difference lies in the information that is used to do the forwarding decision. Routers use Internet Protocol (IP) addresses, and thus operate on layer 3, for the forwarding decisions while switches use Media Access Control (MAC) addresses, and thus operate on layer 2 only. This difference leads to routers needing to run different routing protocols to gather information about the network and to construct routing

and forwarding tables. These protocols also typically are configured manually, which makes installation of a router more cumbersome. Switches on the other hand are self learning devices that configure their forwarding tables automatically. The learning is done by reading the source MAC address of the packets sent through the switch and at the same time noting the port that the packet arrived on. By using this information, the switch quickly learns which MAC addresses reside on which ports and can thus populate its forwarding table with the data. Typically switches are used in smaller networks where less control over the data flows is needed, whereas routers are used in big installation where control is important.

2.2 ASIC

As the name Application Specific Integrated Circuit says, ASICs are circuits that are specifically designed to do a certain predefined set of tasks. Such tasks in a router or switch might be related to QoS, packet marking or scheduling. The common denominator for the typical ASIC task is that it needs to be done at line rate for the incoming packets.

Typically ASICs are used when such performance is needed that it cannot be achieved with regular Central Processing Units (CPUs) nor NPUs. There are some drawbacks to using ASICs though. First of all they are most often not programmable or offer very limited programmability, which makes adding new features such as protocols a very demanding task [10]. Each time a new feature needs to be added, the production of the old ASIC needs to be stopped and an extension to the original chip or, a totally new chip, designed. In the worst case scenario a customer that wants to utilize this upgrade needs to buy a whole new piece of equipment instead of just upgrading software like one would do with the alternatives that offer some sort of programmability. This feature of ASICs adds costs to both the manufacturer and customer in the form of redesigning a whole chip or buying new equipment respectively. Also having to always upgrade equipment instead of software is probably not going to be a big hit with customers looking for long term relationships with their vendors.

Another downside of ASICs is the time required to come up with a working design. Typically the design process for an ASIC, used in a high-speed networking environment, takes somewhere around 12 to 18 months. During this time significant changes and innovations in the protocols used might have been made, making the chip already outdated at the time of release. Also the specification work done before the actual production of an ASIC leaves no room for error. When the specification work is done and production of the chip started, one must assume that design is actually working as it should as it is impossible to make changes to the product

afterwards. The same thing applies for new additional functionality that there might become a need for during the design process [11]. In some cases it will be impossible to add that new feature without some significant changes needed to the original design. Sometimes it is just easier to make a whole new design than trying to redesign an already existing architecture, which leads to a lot of money being basically thrown away. Thus the specification process of the ASIC design is a key factor to the success of the product.

As ASIC design was so expensive, vendors started reusing already working chips in as many products as possible. These so called Application Specific Standard Products (ASSPs) were chips that were designed for other vendors to use in their products, which meant that they could spare some money in their design processes. The ASSPs were later developed into Application Specific Instruction Set Processors (ASIPs) that offer a small amount of programmability either through instruction sets or hardware based on Field Programmable Gate Arrays (FPGAs). Thus the ASIPs are a good way of shortening the design phase and extending product lifetime. Unfortunately ASIPs consume roughly ten times the power of a regular ASIC and do not deliver the same performance. For a long time each function that equipment vendors integrated to their products were implemented on separate ASICs. Although the design process of such a chip was long and expensive, the manufacturing of them was very cheap, which lead to wide adoption of ASICs in network equipment manufacturing. Recent advances in silicon technology have reduced the chip sizes so much that the limiting factor for size is now the required throughput. As the requirements for throughput grow so grows the amount of connector pins required on the chip, which essentially makes a modern NPU chip equal in size to an ASIC, which eventually leads to the prices being almost equal making many manufacturers favor the NPUs.

2.3 Network processors

Network processors or NPUs [10] as they are also commonly known, form a family of highly specialized programmable processors. Their main use case is to optimize packet processing functions in routers, switches and other networking equipment. Packet processing speed is one the key features in such equipment and thus it determines the value of these machines to the customers. The choice of using a general purpose CPU, an NPU or an ASIC when designing networking equipment leads to a debate between the choice of flexibility and performance. Network processing units offer a solution that lies somewhere between the programmability of a CPU and the performance of an ASIC.

Network processors can be categorized in many ways. One way is to divide the

field into platform and peripheral NPUs. Platform NPUs are designed to minimize hardware costs of the final product by including all the necessary packet processing functions in one chip. The peripheral NPUs on the other hand are such that they are optimized for one function only and can thus only be used to extend the feature set of another chip. An example of such a chip could be an IP Security chip used to enhance the processing of such packets. Another way of categorizing NPUs is by their throughput performance to entry-level (1-2 Gbps), mid-level (2-5 Gbps) and high-end (10-100 Gbps) NPUs [11]. A third way of categorizing NPUs is presented in [12]. In this research categorization is done by the parallel processing approach taken, special purpose hardware included in the design, used memory architectures, on chip communications mechanism or the peripherals.

2.3.1 Network processor design

Network processor design consists of many different smaller design choices that need to be made. One of the first is the choice of how to arrange the different Processing Units (PUs). PUs are the parts inside the NPU that do the actual processing such as checksumming and packet manipulation. Figure 2 illustrates some of the most common topologies of PUs. The simplest way of arranging them is to put several identical PUs parallel to each other (1). In this topology each PU is responsible for all the processing needed by the packet. Packets are assigned to the different PUs by a scheduler, which always chooses one that is free. Another topology using identical PUs is where they are arranged in a pipeline (2). In this topology each PU does just one small part of the processing required by a packet. The problem with this architecture is that to get the most out of it, the workloads per PU should be divided so that each PU does the same amount of work. If the amounts of work are not the same, one PU might stall the whole pipeline as other PUs are waiting for it to finish. This problem can be solved by inserting buffers between the PUs. Another pipelined topology features PUs that all are specialized for a certain function (3). This clearly is an advantage over the pipeline of identical PUs as each PU is optimized for its own task, which improves efficiency. An improvement to the pipeline of identical PUs is one where several such pipelines are arranged in parallel (4). This topology improves performance by reducing the overall probability of stall, as each new packet is scheduled to be processed by a pipeline that is idle. For applications requiring very high throughput a pipeline of parallel PUs can be constructed (5). In this type of design there are several identical PUs at each stage of the pipeline, but each stage contains PUs that are optimized for a certain task. The packet that is being processed is always forwarded to an idle PU.

Other major issues faced during the design of NPUs are the choices of Instruction Set Architecture (ISA), memory architecture as well as internal and external

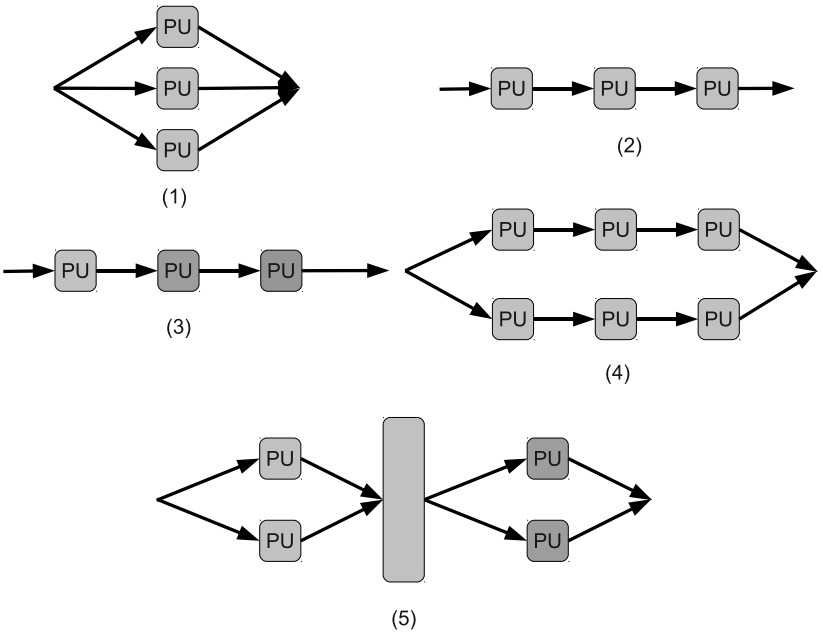

Figure 2: Different NPU PU topologies.

buses and interfaces. The ISA choice mainly relates to choosing how specific the used instructions should be regarding the tasks at hand. If for example an ISA containing specific instructions related to different protocols is chosen, it might be that a protocol is updated and suddenly a large part of the instructions are useless. On the other hand a very general instruction set might require a lot more effort in programming as a typical action such as checksumming might require several instructions to be completed. A general issue when designing any microprocessor is the one considering size of the package that the actual chip resides in. Internal buses often work in parallel and might be up to several hundred bits wide. Thus they take a lot of space inside the chip. For NPUs that don't need the higher throughputs offered by the parallel buses a Network on Chip (NoC) can be built that connects all the elements. Also the choice between using memory on the chip or memory outside of the chip affects the area. Processing packets at line rate requires a lot from the memory, in terms of latency and bandwidth, so using memory on the chip would be the optimal choice but at the same time the size of the chip would grow. An important decision is also the one considering what kind of external interfaces are needed. External interfaces are used for example to connect the NPU to the physical layer chips, co-processors and switch fabrics.

2.3.2 Pros and cons of network processors

The benefits of using NPUs are many. Developing an ASICs chip is very cumbersome and takes a lot of time and effort. Instead by using an NPU only the software doing

all the packet processing needs to be developed, which can be done in a much shorter duration of time. The programmability of the NPUs enables new features to be added to the equipment after it has been deployed by customers. This means that the life cycle of the product is extended, which in turn adds more opportunities for the producer of selling it. That is obviously good for your business and creates an opportunity for extended relations to customers, because they are more willing to invest in a product that has a longer life cycle. The programmability that enables adding of new features is obviously also one of the key benefits as the new features can be added without costly hardware replacements. Finally using NPUs frees resources to other activities inside the companies. These can be used to improve things such as business relations and management or to add necessary features and functionality to the devices manufactured.

The NPUs offer good solution to many problems but are by no means perfect. One of the challenges is programming of the NPUs [13]. As each processor has its own proprietary architecture there is no single unified way of programming them; some manufacturers strive to hide all programming details while others give full control to the programmer making things a lot more complicated. These architectural choices lead to difficulties in estimating the software development costs and to longer than expected software development projects. Integrating the NPUs to ones design has also been a big issue. That is why manufacturers have created their own ecosystems around their different networking chips to make integration as easy as possible. Some implementation agreements have also been proposed by the former Network Processing Forum (NPF) which is now part of the Optical Internetworking Forum (OIF) [14].

2.3.3 Manufacturers

Some of the bigger NPU vendors are EZchip Technologies [15], PMC-Sierra [16] and LSI Corporation [17]. EZchip is the leading manufacturer of NPUs that are capable of throughputs over 10 Gbps. Their leading product at the moment is the NP-4 which was the first NPU to achieve a throughput of 100 Gbps. EZchip is currently working on an NPU capable of 200 Gbps. PMC-Sierra on the other hand is a company that is focusing its efforts on devices used in the access parts of networks instead of high-end core devices that EZchip targets its products for. PMC-Sierra produces the WinPath line of NPUs, which can achieve throughput rates of tens of Gbps. LSI has chosen to combine regular PowerPC cores with an NPU in their Axxia Communication Processor, which is capable of throughputs in the order of 20 Gbps. Other notable vendors of NPUs are Broadcom [18], Xelerated [19] and Intel out of which Intel has stopped developing new NPUs but are still selling their old models. At the moment all big equipment vendors such as Cisco and Juniper are

using NPUs from other manufacturers in their products even though they also have their own NPU designs. The chips from other vendors are mainly used in the lower end products. The NPU market saw a 68% growth in revenue between the years 2006 and 2010 and is expected to continue growing through 2015.[20]

2.4 x86 architecture

The history of the x86 architecture dates back to the year 1978 when Intel released the 8086 processor. The 16-bit 8086 was the successor of the, at the time, very successful 8-bit 8080 processor [21]. This processor was the starting point for an ISA that would be updated several times during the coming years, and finally become the most successful ISA of them all. The success of the 8086 is strongly related to the success of IBM's first PC that included a scaled down version of the 8086. After the success of IBM, other manufacturers started making their own PCs from similar components, which further increased the success of x86. The x86 name that is commonly used for the ISA actually comes from the number codes that Intel used to name the processors.

The first major upgrades for the x86 architecture came in 1982 and 1986 when the Intel 80286 and 80386 processors were released. The 80386 was the more significant of the upgrades as it extended the x86 architecture to 32 bits. Until 1997 the ISA was held quite stable as the 80486 and Pentium processors only added four new instructions to it. Through 1997 to 2003 several hundred new instruction were added to x86 in the forms of Multi Media Extensions (MMX) and Streaming SIMD (SSE) instructions. Until the year 2003 Intel was the only company that had modified and extended the x86 ISA. In this year Advanced Micro Devices (AMD) introduced the AMD64 architecture that further extended the x86 architecture to 64 bits. Intel did not first support the AMD64 architecture, but after a year they released their own version of it. Since the release of the 64 bit version the x86 architecture has been updated quite regularly with new versions of SSE and lately with the Advanced Vector Extensions.

The benefit with the x86 ISA is that it has retained backwards compatibility. A program written for the first 8086 processor will still run on the newer generation processors without modification [22]. This backwards compatibility also has a drawback in that the instruction set is very large and complex. On average one new instruction has been added to the design for each month that the architecture has existed.

2.4.1 Nehalem

The first processors using Intel's [23] new Nehalem microarchitecture [24] were released in early 2009. The architecture was designed especially with power consumption in mind. Each new feature added to the processor should add at least one percent of performance gain with at most a three percent increase in power cost, to be included in the final processor. The first Nehalem processors were manufactured using a 45 nm Hi-k metal gate process, which was later upgraded to 32 nm. The lineup of processors using this architecture includes both server and desktop processors. The server and desktop processors are sold under the Xeon and Core i3/i5/i7 brands respectively.

The Nehalem processors all have between four and ten cores each. Thanks to Hyper-Threading Technology each core can process two threads simultaneously. Thus, for the operating system, it looks as if there would be double the amount of cores on the processor. Hyper-Threading can offer as much as a 20-30 percent boost in performance with minimal increase in power costs. Each core has its own level 1 and level 2 caches of 64 and 256 kilobytes respectively. In addition to these, a shared level 3 cache of 12 megabytes is available. The shared cache is inclusive, which means that if the wanted data cannot be found in the level 3 cache it will not reside in any of the higher level caches of other processors. Figure 3 illustrates the cache hierarchy of a quad core Nehalem processor.

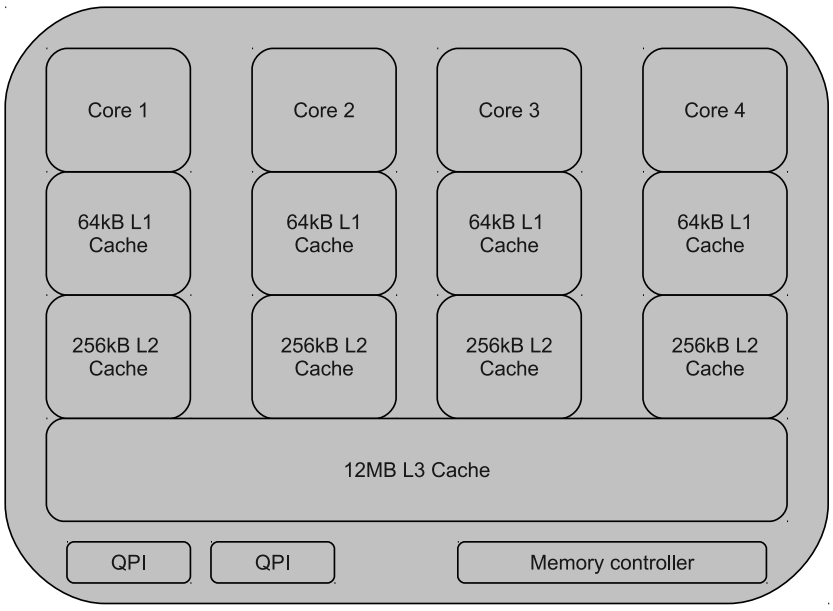

Figure 3: Organization of a quad core Nehalem processor.

One of the key innovations in the Nehalem microarchitecture is the Turbo Boost [25] technology. This technology enables one or more of the processor cores to run

at a higher operating frequency, assuming that predefined constraints set on power consumption, temperature and current draw of the processor are not exceeded. The amount of boost available at each moment depends on the previously mentioned constraints but also on the amount of cores active. When the Turbo Boost is activated it is done in steps of 133,33 Mhz, where the amount of steps varies with the processor type. The processor can stay in the boosted state for as long as the constraints are not exceeded. Research [26] conducted on Turbo Boost show a 6% reduction in average execution time of applications with the feature enabled.

Older system architectures have housed the memory controller in the northbridge that has been connected to the processor by means of the Front Side Bus (FSB). The northbridge is the main logic chip on the motherboard and typically functions as an endpoint to PCI Express (PCIe) connections in addition to housing the memory controller. It is also connected to the southbridge chip that handles most other Input/Output (I/O) related tasks such as USB and SATA ports. A key innovation in the Nehalem microarchitecture is that each processor has its own memory controller. In that way access to memory is not limited by the capacity of the front side bus, as it was in the older microarchitectures. Now memory can be accessed directly without having to wait for other devices to finish their transmission on the bus first. This invention also lead to a whole new system architecture design, in which the old shared front side bus technology was replaced with a new point to point bus technology called the QuickPath Interconnect (QPI) [27].

2.4.2 QuickPath Interconnect

Before the release of the Nehalem system architecture all communications between the processors, memory and other devices went through a shared bus called FSB. This meant that all the peripheral devices had to compete for access to the bus and the memory. Also as the buses were transferring up to 128 bits at a time using around 150 wires and running synchronously at very high frequencies there were some issues related to electrical constraints. Later, to further increase the capacity of the buses the shared bus architecture evolved into dual independent buses between the northbridge and processors. At this time the memory controller was still located at the northbridge. This lead to an architecture with dedicated buses for each processor which finally lead to the QPI architecture. Figure 4 shows a comparison between the two technologies.

In difference to the old shared bus, the QPI architecture consists of a set of point to point links similarly to PCIe. The data in QPI is sent in packets, which are split into parts that are sent in parallel on multiple lanes. The QPI protocol architecture consists of five layers namely the physical, link, routing, transport and protocol layers. The physical layer consists of the actual wires that make up the link. Each

link consists of twenty signal pairs whereas two links constitute a connection between two QPI connected devices. In addition each link has one pair of wires dedicated for a clock signal. Not all the wires have to be used for the link to work. The QPI specification states that the link can also be used at half and quarter widths. Thus it is able to transfer data at rates of 5, 10 and 20 bits at a time. The maximum transfer rate for a QPI link is thus 6.4 Gigatransfers per second (GT/s), comparing to 1.6 GT/s for a legacy front side bus. For the QPI bus these transfer speeds equal to bit rates of 32, 64 and 128 Gbps.

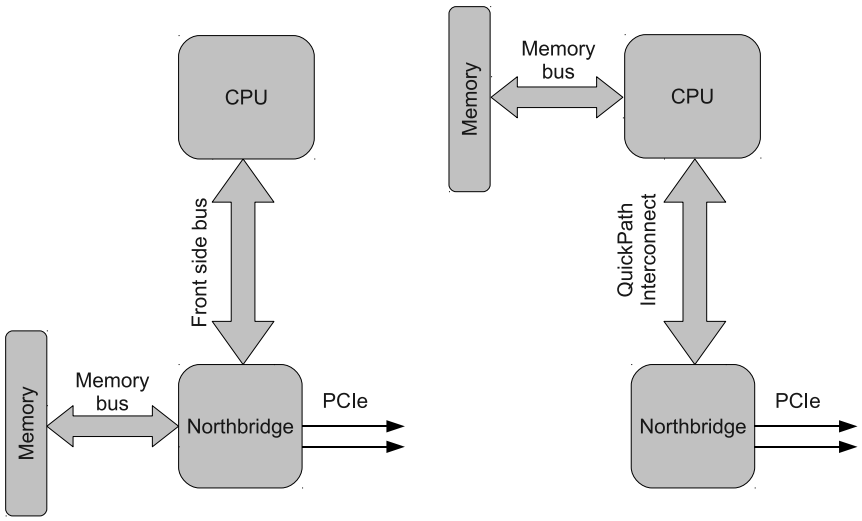

Figure 4: Comparison of front side bus and QPI architecture.

The link layer takes care of checksumming the data between two QPI devices and thus guarantees reliable data transfer, provides flow control between them and abstracts the physical layer. The link layer transmits data in 80 bit flow control units out of which 8 bits belong to the checksum. If a checksum fails a retransmission is requested. Flow control works by a credit system where each sender is given a certain amount of credits at the beginning. Each time a flow control unit is sent the sender decrements its credits by one. Whenever the receiver has processed a received flow control unit, it sends one more credit back to the sender. If the sender is out of credits it will wait for new ones before sending more flow control units. In that way the sender will never exhaust the receiver with an overwhelming amount of packets at one time.

The routing layer takes care of forwarding packets to the correct devices according to routing tables defined by firmware. At the moment the transport layer is not implemented but its task is to guarantee end-to-end transmission reliability.

The protocol layer transmits packets and houses the cache coherency protocol that maintains the caches of the system. All in all the QPI is able to transmit some 12,8 Gigabytes per second (GB/s) of raw data per direction meaning that the bandwidth of one connection is 25,6 GB/s. In the future QPI is expected to deliver even higher bandwidths that are needed with the ever increasing system performances [28]. Due to the layered design of QPI these changes will be easy to implement as they only affect one layer. The other layers need not know about the changes as the interfaces between them will stay the same.

2.4.3 Future Intel x86 architectures: Sandy Bridge and Ivy bridge

When the writing of this thesis started Nehalem was still the number one microarchitecture from Intel. Since then the situation has changed as Intel introduced processors using the Sandy Bridge [29, 30] microarchitecture, and is about to release the first processors using its successor Ivy Bridge [31]. Sandy bridge is a totally redesigned architecture compared to Nehalem that used many features of older microarchitectures. One of the biggest differences is that Sandy Bridge is from the beginning manufactured using a 32nm process, which enables better performance for less power consumed. Nehalem integrated the memory controller to the processor but Sandy Bridge takes integration one step further. The new processors in addition to the integrated memory controller include the controller for PCIe buses and a Graphics Processing Unit (GPU). By doing this the northbridge and GPU chips can be removed from the motherboard. The southbridge chip is still being used as a hub for other I/O buses.

One major feature in Sandy Bridge is that it is able to do two memory loads or stores or one load and one store simultaneously, compared to only one load and one store simultaneously in the Nehalem microarchitecture. Also the handling of so called micro operations has been improved by adding cache for them. Floating point calculation performance has been increased by adding new Advanced Vector Extensions (AVX) instructions. The components that reside inside the processor , that is, the processing cores, memory controller, caches, PCIe controller and GPU, are connected by a ring based interconnect instead of a crossbar that was used in previous microarchitectures. This makes designing new processors with different numbers components inside them easier as the ring just needs to be extended instead of designing a new crossbar. The ring interconnect uses an enhanced version of QPI for communications between the components. Also the Turbo Boost and power management features have been improved in Sandy Bridge.

The Ivy Bridge microarchitecture is an upgraded version of the Sandy Bridge one. It is manufactured using a 22nm process, which again increases performance while reducing power consumption. The integrated GPU has also been updated.

Notable other features are a digital random number generator that is designed to be standards compliant, further improvements in power management and overclocking support.

2.5 PCI-Express

The different generations of Peripheral Component Interconnect (PCI) buses are used to connect different I/O devices such as graphics cards and network adapters to a computer. The original PCI bus specification was first released in 1992 [32] by the PCI Special Interest Group (PCI-SIG). This version offered a data rate of 133 MB/s using a 33 MHz clock and a 32 bit wide bus that was later extended to 532 MB/s by doubling the clockrate and width of the bus. These buses then evolved to the PCI-X bus that offered an increase in bandwidth up to 1 GB/s. Looking at these figures it is easy to see why a replacement for the older PCI technologies was needed. Take for example the dual port 10 Gbps Ethernet adapters that were used for the experiments in this thesis. Each port is able to handle full duplex traffic which makes the total of 40 Gbps for each adapter. This is five times more than what the the PCI-X bus was capable of and clearly illustrates the need for a new faster interconnect. In addition to the higher bandwidth requirements the new PCIe bus was designed such that it would become a solution used everywhere from regular desktops and high end servers to mobile and embedded devices [33]. It was also designed to support scalable performance and be compatible with the already existing PCI software architecture. On the other hand PCI-SIG did not strive to create a new processor or memory interconnect.

Just like QPI the PCIe bus topology is built up of a set of point to point links instead of one big shared bus that the older variants of PCI used. All the PCIe links are connected to a switch that can either be a separate component or as it often is nowadays directly integrated to the northbridge or some other chipset. The PCIe architecture consists of five layers. The lowest layer in the stack is the physical layer that transmits data on the actual wires. Each PCIe link consists of at least two unidirectional pairs of wires that constitute what in PCIe terminology is called a lane. A physical link between two PCIe devices typically consists of several lanes. At the moment link widths of x1, x2, x4, x8, x12, x16 and x32 lanes are supported by the standard. Figure 5 clarifies the differences between links, lanes and wires. In PCIe 1.0 one lane works at 2,5 GT/s per direction, which was raised to 5 GT/s in 2.0 and 8 GT/s in 3.0 [34]. These rates correspond to raw bitrates of 5, 10 and 20 Gbps per lane respectively. Versions 1.0 and 2.0 use 8b/10b encoding for transmissions so a 20% overhead should be taken into account.

The next layer is the data link layer, which is responsible for reliable data delivery.

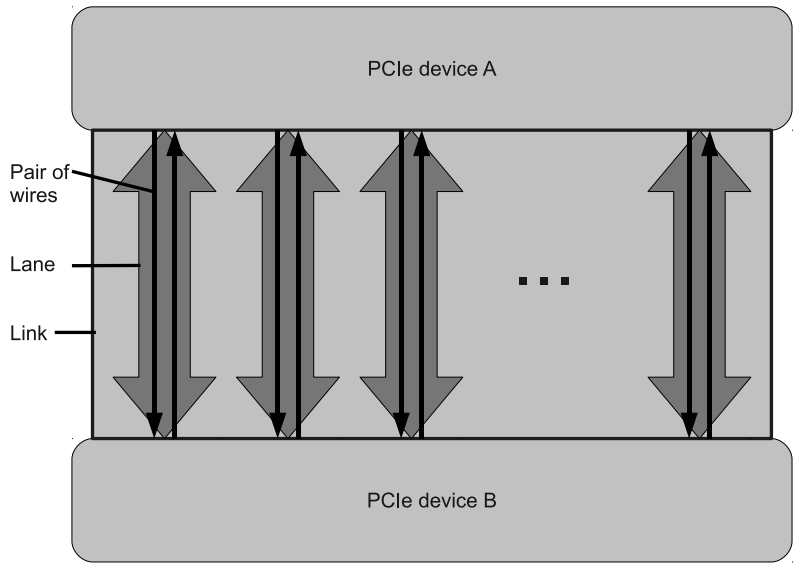

Figure 5: PCI-Express link architecture.

This is achieved by using a checksum and sequence numbers. Flow control is achieved by a similar credit based system as in QPI, where the receiver sends credits to the sender when buffers are available, thus enabling the sender to send. Next up is the transaction layer that creates request packets depending on the reads and writes done by the software layer. It also routes the received responses to the correct requester. The transaction layer also handles the Message space address space. This address space has a great importance in the functioning of PCIe as it is used to transmit interrupts and other side band signals that in previous generations of PCI used their own wires. The topmost layer in the PCIe stack is the software layer which in addition to generating reads and writes enables the OS to discover all the connected PCIe devices. Thus the OS can allocate the required resources to them.

2.6 Memory hierarchy and technologies

Different types of memories and caches are needed in computers to speed up execution of programs, as always reading and writing all data from/to a hard drive would slow down execution considerably. Today's computers use a hierarchy of different types of memories typically consisting of fast and small caches directly connected to the processor, bigger main memory connected to the processor by means of a separate memory bus and finally some sort of hard drive that is used to store all permanent data [21]. When the processor needs a piece of data such as an integer to complete a certain instruction it will first look for the data in its cache. In case the data is not found from the cache, it will be fetched from the main memory to

the cache and finally to the processor. The next time that the processor needs the data it will already be in the cache and can thus be accessed faster than fetching it from the main memory. The first condition is called a cache miss and the second a cache hit. The rates at which these happen are important measures of system performance. Similarly when the processor writes data it first writes it to the cache from where it is then written to the actual memory location.

The previous was an oversimplified version of what caches are and how they work. Many issues need to be overcome when designing caches for example the trade-offs between cache size, hit rate and speed. A bigger cache will generate more hits but be slower than a smaller cache that in turn does not generate as many hits. Another issue arises from the modern multicore CPUs. If two cores have cached the same memory location and if one or both of the processors modify the value, how will the other be notified of this and which value will be written to memory. Similarly to regular caches the main memory functions as a cache for the hard drive. Typically this means that when a program is loaded on a computer, it is loaded to main memory for the time of execution. When the program finishes, the modified data is then written back to the hard drive. Modern microarchitectures use a hierarchy of caches containing several levels. Intel's Nehalem for example has three levels of caches where each core has an individual level 1 and level 2 cache and all cores share a level 3 cache.

Cache memory is typically of the Static Random Access Memory (SRAM) type because of its fast access time and low power consumption compared to other technologies. The downsides of SRAM are its higher price and and larger area required to store a bit. Main memory on the other hand is built of Synchronous DRAM (SDRAM) type of memory and nowadays mainly of the Double Data Rate type three (DDR3) variant of it. Compared to regular Dynamic Random Access Memory (DRAM), SDRAM works faster due to its synchronous nature. This also makes for a remarkably simpler memory controller design [35]. The DDR3 memory modules are connected to the processor using a 64 bit wide memory bus. This bus can work at frequencies ranging from 400 to 1066 MHz. DDR3 SDRAM doubles the rate of data transfer compared to regular SDRAM by transferring data on both the rising and falling edges of the memory clock signal which equals to memory bus transfer rates of 800 to 2132 MT/s. In bits per second this would equal to some 51,2 to 136,5 Gbps.

In order to further increase memory performance modern computer systems use triple channel memory [36]. This technology is enabled by widening the memory bus to 192 bits (= 3*64). The memory modules are still the same that are used in single channel operations, but a multiple of three modules needs to be installed in the system. Each of the modules then gets its dedicated 64 bit memory channel,

which in theory triples the memory bandwidth seen by the system. Non Uniform Memory Access (NUMA) is a memory architecture that is used in computers consisting of several CPUs. In the NUMA architecture each processor has its own local memory and integrated memory controller. The entity consisting of the processor and its local memory is called a NUMA node. Figure 6 illustrates a typical NUMA architecture consisting of two nodes. Modern operating systems strive to allocate memory such that, each process gets memory from the same node that the CPU running it belongs to. Still situations may arise where some data may be needed from the remote memory located in the other node than the CPU, which is why the nodes are connected to each other. It is also a requirement of the x86 architecture that all CPUs have access to all memory in the system. Intel uses the QPI bus introduced in Section 2.4.2 to connect NUMA nodes.

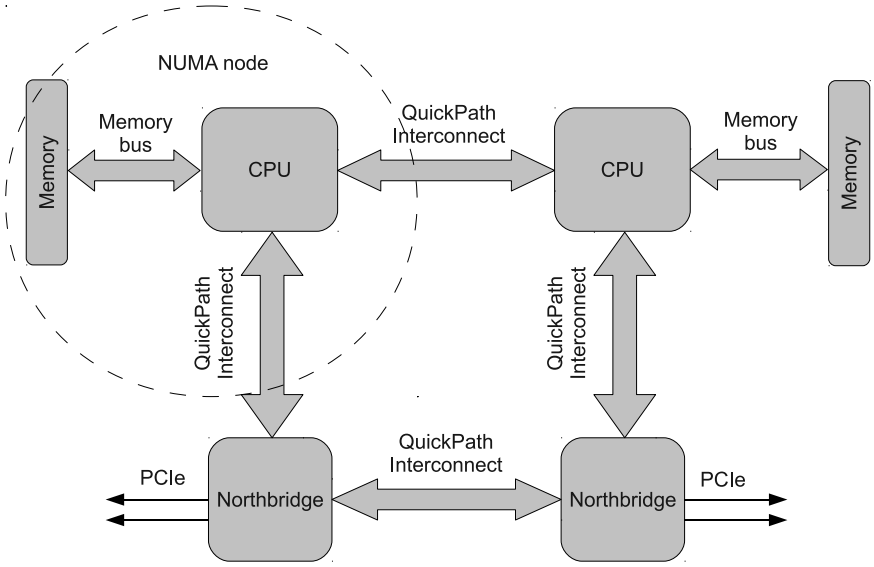

Figure 6: NUMA architecture consisting of two nodes.

2.7 x86 in networking applications

The main motivation for using COTS hardware such as any regular x86 processor in networking is the simple programmability of such a device. In case modifications are needed to the packet forwarding, only the software needs to be upgraded. This process is vastly shorter than for example the development of a totally new ASIC or NPU. A software upgrade can be coded in weeks while the design of a new chip will surely take at least several months. The programmability also extends product lifetime as hardware upgrades are not needed all the time. The problem with the x86 CPUs is that they are designed for general purpose computing, which

networking is not. This means that the CPUs will be processing many instructions that are not related to the actual packet processing. This shortage of processing power is becoming more obvious as requirements for deep packet inspection are becoming more common. Deep packet inspection is a type of processing where some information needs to be extracted from the higher application layer protocols. This could be for example be an Uniform Resource Locator (URL) from the Hypertext Transfer Protocol (HTTP) header. Deep packet processing is very resource intensive and it is expected that COTS hardware will not be able to do it at the very high speeds required today [10].

There are also problems with networking on x86 hardware when considering the more technical aspects of the processors. A typical modern processor will process billions of instructions per second due to a pipelined architecture and a clock frequency of some gigahertz. This is not the whole truth as the processor needs to access memory in order to store results and fetch data for forthcoming operations. Memory accesses cannot be completed at the speed the processor executes instructions, which has lead to complicated architectures consisting of multiple levels of caches and memory that all need to be synchronized. This leads to a situation where the processor pipeline is empty and severely underutilized but at the same time the system cannot handle the offered traffic because of the memory system access time. Another problem is that statistical properties of network traffic have not been taken into account in the design of the cache subsystem. Thus the cache system does not work as it is supposed to, which will eventually slow down the processor. Finally a huge problem is that packet processing requires some relatively specific instructions for doing bit level operations on the data. Typically such instructions are not present in commodity processors, which means that the processor needs to execute several instructions to achieve something that for example an NPU can do with only one instruction.

2.8 Optimizing hardware performance

In general one can think of optimizing performance of a computer system from two different viewpoints; hardware and software. The simplest way to get better performance from either of these is to directly upgrade to newer and better versions of the hardware or software. If one does not want to do that, different settings of both hardware and software can be changed. Hardware settings can typically be changed either from the Basic Input/Output System (BIOS) system or through the driver for the particular device. Software on the other hand typically has its own settings. The OS also greatly affects the performance of a computer as it keeps the package together and thus affects all programs running. Its performance can for example

be increased by tuning different kernel parameters or by compiling one's own kernel with all the unnecessary features disabled. The software side of performance tuning will be covered in Section 3.7. [37]

2.8.1 Optimization cycle

Before starting to do any changes one should somehow measure the performance of the system and at the same time try to identify some bottlenecks. By doing this a baseline can be established, that then later on can be used to compare the effects of changes made to the system. When all these steps are combined together and repeated after each other, an optimization cycle or workflow is created. The cycle is illustrated in Figure 7. The metric that is used to define the baseline should be chosen based on the application that the system is used for. For a software router these metrics could be for example throughput in packets per second, the latency of the system or errors in received packets. Other typical metrics that are used are ones related to CPU, memory and I/O-device usage. Such metrics might be for example CPU utilization, times spent processing user or kernel processes, free memory, memory activity, I/O device queue length and transfers per second.

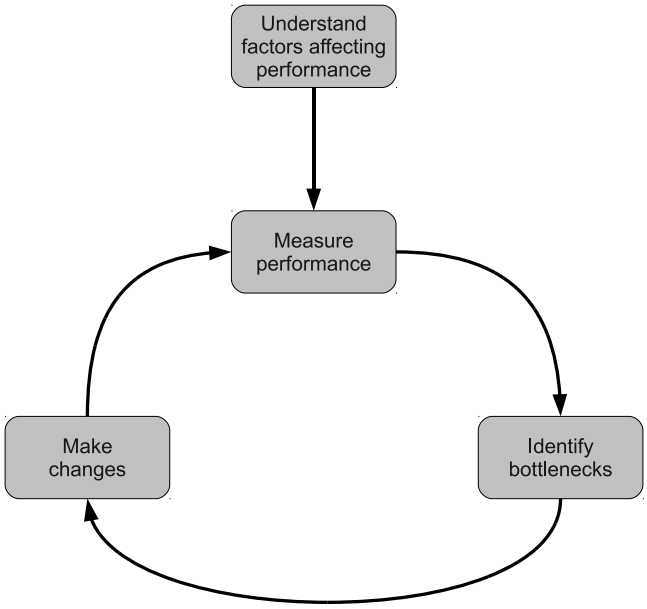

Figure 7: System performance optimization cycle.

When the baseline has been established and some of the bottlenecks identified, one can start tuning different parameters. This should be done one parameter at a time, after which the effects of the change should be tested. Only after it has been confirmed that the change has a positive effect should one make another change. Typically one can define the bottleneck with accuracy of the subsystem that is

causing the problem, after which it is easy to start making changes. Typically the system would for example be divided to the CPU, memory, disk and network subsystems, which all have their own guidelines to tuning. Performance of the disk subsystem is not critical to the performance of a system under a software routing or switching workload, which is why no tuning guidelines will be given for it.

2.8.2 Optimization using BIOS

The BIOS system is used to make changes in how the motherboard of the system interacts with the components connected to it. It offers several options to tune for example the CPU, memory, different buses, various options related to the northbridge, different boot and power management options. The amount of configurable parameters is huge and thus only the ones that are deemed to have any significance on the packet forwarding performance of the system will be covered. The effects of the settings will be covered in Section 4.5.

One of the most important parameters that directly affects the systems performance is the *CPU clock ratio*. It sets the ratio between the clock generator of the system and the CPU's clock and thus lets the user choose the speed that the CPU is running at. Another important parameter is the one controlling *simultaneous multithreading* (also known as HyperThreading on Intel processors). By enabling it, each physical processor core gets a virtual sibling that is able to do processing similarly to the physical core. This essentially doubles the amount of threads that can be executed by the processor at one time and may improve performance greatly. The amount of active processor cores can also be changed directly. Intel *Turbo Boost* technology (Section 2.4.1) can also be enabled through BIOS. [38]

The caching system is very important in speeding up execution of programs. The way that data is stored to caches can be changed by enabling or disabling *Hardware Prefetch* and *Adjacent Cache Line Prefetch*. Hardware Prefetch transfers instructions and data from the main memory to the L2 cache that the CPU predicts that are needed in the future, by analyzing the code that is executed. Adjacent Cache Line Prefetch on the other hand just fetches two adjacent cache lines for each request instead of only one. Other notable processor optimizations are *Enhanced Intel SpeedStep Technology (EIST)* and different *C state technologies*. EIST is, as the name says, a technology that allows automatic adjusting of the processor's voltage and core frequency depending on the system load. C state technologies on the other hand relate to shutting or slowing down single processor cores that have been idle for some time. By disabling both of these features one gets rid of the delays that are caused by the processor moving from state to another. In theory this should also increase performance.

Both QPI and memory *bus bandwidths* can be directly controlled through BIOS.

The speed they are running on has a big impact on performance as all the devices communicate to each other using these buses. The *maximum payload size* of the PCIe bus can also be adjusted. Adjusting this value is important as some devices perform better by using smaller values. Memory *channel interleaving* is also an interesting option when thinking about performance improvements. When using all three channels of the memory controller in an interleaved way and trying to access a contiguous block of memory, the first 64 bits will be accessed on channel one, the second 64 bits on channel two and so on. Using memory in this way reduces access time and should also increase performance. If interleaving is disabled all the accesses would be queued on channel one, which would cause a delay in the access and thus reduce performance.

Other interesting features are Intel's *I/O Acceleration Technology (I/OAT)* and features related to virtualization such as *VT-d*. I/OAT promises to address the most common bottlenecks found in packet processing and thus makes for an interesting feature. The different virtualization solutions such as VT-d are not interesting for the performance that they might add, but for the overhead that they contribute. By disabling these features one could expect to see some positive effects in packet processing performance.

2.9 Summary

This chapter introduced the internal workings of a router and the different components that one can be constructed of. A deeper look at the Nehalem microarchitecture by Intel was also given. Typically high end routers that are used by large enterprises and operators are constructed using ASICs or NPUs. They both offer a lot more performance wise than for example a router operating on x86 hardware. The downside of constructing a router or switch using ASICs and NPUs is that they are not as flexible to changes as just using software for packet forwarding. ASICs offer limited programmability while the NPUs can be programmed but there might be some limitations in the ISA compared to using an x86 processor.

Intel's Nehalem was the choice of microarchitecture for this thesis although Sandy Bridge processors have already been released and Ivy Bridge is nearing its release. Nehalem was the first microarchitecture to integrate the memory controller on the processor chip. By doing this Intel got rid of the FSB and replaced it with the point to point QPI bus architecture. The QPI bus is also used in the NUMA architecture as an interconnection between the NUMA nodes. This chapter was concluded with a look at the process of optimizing hardware performance by making changes to the BIOS settings. Table 1 summarizes the BIOS settings that were presented in this section.

Table 1: BIOS settings that were thought to have an effect on forwarding performance.

Parameter	**Effect**
CPU clock ratio	Controls the speed of the CPU.
Simultaneous multithreading	Enables the processor cores to have virtual siblings.
Turbo Boost technology	Lets some of the processor cores run at higher frequencies.
Hardware Prefetch	The CPU predicts needed instructions and transfers them to L2 cache.
Adjacent Cache Line Prefetch	Two cache lines are fetched for each request instead of just one.
EIST	Allows the processor to control voltage and core frequency based on system load.
C state technologies	Allows the processor to shut down cores that have been idle for some time.
Bus bandwidths	Control the amount of data the bus in question can transfer.
Memory channel interleaving	Controls how the memory is accessed.
I/OAT	Removes some bottlenecks related to packet processing.
VT-d	Enables some features that are needed for virtualization but not packet processing.

3 The Linux operating system

In addition to the hardware found inside a computer the operating system plays a major role in making the system work, as it basically glues everything together. This chapter covers the networking aspects and other relevant features of the Linux kernel that affect the performance in switching and routing situations.

3.1 History

The history of Linux dates back to 1969 when Ken Thompson and Dennis Ritchie in cooperation with researchers at AT&T Bell Laboratories developed the Unix operating system [39]. In the 1970s Unix was licensed to several major companies and institutions. One of them was the computer science department at University of California, Berkeley, that started releasing their own version of Unix namely Berkeley Software Distribution (BSD). In 1983 they released version 4.2 of BSD, which included implementations of the IP and Transport Control Protocol (TCP) protocols, that are still used today and have become the cornerstone of the Internet.

The foundation of Linux was laid at Helsinki University by Linus Torvalds in 1991. At first Torvalds was only going to write a terminal emulator so that he could gain access to the university's Unix server and so that he would become familiar with the architecture of Intel's 80386 processor [40]. The motivation for Torvalds to do his own program was that he was not satisfied with how terminal emulation was done in his OS of choice at that time, MINIX [41]. Through the years the project kept growing from terminal emulator to OS and finally Linux version 1.0 was released in March 1994. At this time several other persons besides Torvalds had contributed to Linux. The most important part of the OS is the kernel that controls all the hardware devices and program execution. The kernel is also the part where all the networking code resides. To be able to do something with the kernel one needs applications such as text editors and other around it. One can choose and install all the applications by themselves, but there are also the so called distributions such as Ubuntu or Debian that include a set of applications chosen by the distribution maker to make one sensible collection out of the millions of programs.

3.2 Linux network stack

When a packet is received or transmitted on a Linux host, it travels through two major entities in the kernel. These are the Network Interface Card (NIC) driver and the actual network stack in the kernel. Both of these parts consist of thousands of lines of code and together form a very complicated system. It would require several books to cover all of this, which is why only quite a brief review of the most critical

functions for both packet reception and transmission will be given. The driver is the part that interfaces the NIC hardware to the kernel. At startup it initializes the device and registers the hardware to the kernel and thus lets it know that such a device exists and can be used [42]. The driver also initializes the reception and transmission rings that the NIC uses to store packets before they are moved further up to the actual network stack.

3.2.1 Packet reception

When a multiqueue NIC receives a packet it first uses Receive-Side Scaling (RSS) (presented in Section 3.5) or some other technique to direct the packet to a certain queue. All the queues have their own Interrupt Request (IRQ) (also commonly just called interrupt) that is used to signal the processor that the NIC needs attention. By means of IRQ affinity (Section 3.6) these can be attached to a certain processor core so that one core takes care of the interrupts of one queue, which evens out the load on the system. When the processor receives the interrupt, it notifies the kernel of which interrupt it has received. Based on this information the kernel will call the interrupt handler function that has been registered for this particular interrupt. The registration of the handler is done by the driver when the NIC is initialized. The interrupt handler routine is kept as short as possible, because during its execution all other interrupts are disabled, which could lead to an increase in system response time for other events requiring IRQs such as mouse clicks or disk reads. [43]

The aim of the interrupt handler is to move the packet from the hardware queue of the NIC to the reception ring that the driver has allocated for it from the system memory. The queues being called rings refers to that, the driver always allocates a certain amount of buffers for incoming packets. These buffers are then reused in a circular fashion. When a packet is transfered away from the ring the buffer is marked as free and can be reused when the ring has again rotated to that position. When the packet has been moved to the ring the interrupt handler then notifies the kernel that a packet has been received. This can be done in two ways depending on if the driver supports Linux New API (NAPI) or not. NAPI compatible drivers add the NIC to a polling list and immediately schedule the NET_RX_SOFTIRQ software interrupt. Non-NAPI compatible drivers on the other hand move the packet directly to a special backlog queue, that is unique for each processor, and move the special backlog device to the polling list after which the software interrupt is scheduled. NAPI will be covered in more detail in Section 3.4. The software interrupt is the part of the network stack that does the rest of the packet processing work. The idea behind software interrupts is that by moving most of the processing away from the actual hardware interrupt handler the OS does not get stuck in processing only interrupts [44]. This could be the case for example under high network load and

would result in the system getting jammed and no packets flowing had it not been for the software interrupt.

When the OS scheduler decides that it is time to run the NET_RX_SOFTIRQ it calls the handler function that has been registered for the software interrupt in question. This handler selects the first device in the NAPI polling list and calls its poll method. The poll methods for the NICs are defined in the drivers and registered to the kernel at initialization. The task of these methods is to pull the packets from the NIC's reception ring and pass them to the kernel network stack. When the poll method has fetched a packet, it will then call a function on the kernel side, that finally moves the packet away from the driver's domain and into the kernel. This function then checks the packet type and based on that decides if the packet should for example be forwarded to the switching system or forwarded to some protocol handler on the upper layers, such as the one for the IP protocol.

Counters and timers are used to make packet reception fair at the interface and software interrupt levels so that one interface or software IRQ does not steal all the resources. Each interface gets a weight, which is the maximum number of packets the polling function can pull from each device at one time. If there are more packets in the NIC's reception ring than what can be pulled at one time, the device will be again added to the polling list. In addition to the weight counter, fairness at the software interrupt level is achieved by a budget counter that tells how many packets the software interrupt can process during each run. A timer is also used at this level. It ends the execution of the software interrupt in case that it reaches its end before the budget counter is exceeded. In this case the software interrupt is also rescheduled. Figure 8 shows a basic flowchart of the packet reception process.

3.2.2 Packet transmission

As with packet reception, transmission is also a complex process. When some entity inside the operating system, for example the bridging part, needs to send a packet, it first has to create the packet and find out on which interface it is to be sent on. When this has been done, the sending entity will notify the kernel of its intentions by calling the function responsible for packet sending. The kernel then takes control of the packet and queues it to its own output queue. If the device does not have an output queue, which is the case for virtual devices such as the loopback interface, the packet will immediately be sent to that device. On the other hand when queuing is supported, the function first checks that the queue is active. When this has been confirmed it checks if there are other packets in the queue, in which case it queues the frame and calls a function that is responsible for emptying the queue and moving the packets over to the transmission rings of the NIC. If the queue is empty the packet will be immediately moved to the NIC's transmission rings by calling a function

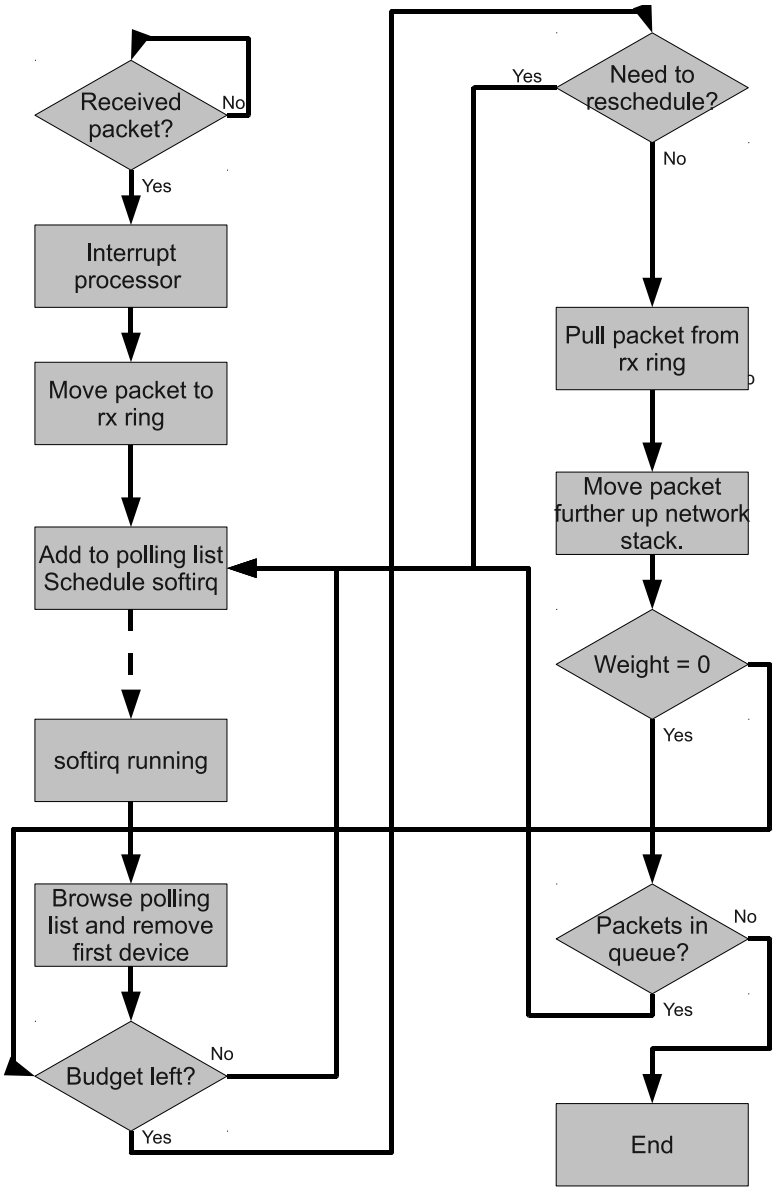

Figure 8: Basic flowchart of the packet reception process.

defined in the driver.

The function that is responsible for transferring packets from the kernel queues to the NIC transmission rings will run for as long as there are packets in the kernel queue, until a preset timer expires or the scheduler decides that it is time to run another program. When the timer expires or the scheduler intervenes, the NET_TX_SOFTIRQ will be scheduled for execution. Finally when the software interrupt is scheduled to run, it will start by freeing buffers that have been used to store packets that have already been transmitted. After this it will start to browse the processors output queue and, by using the same function as mentioned above, move packets to the transmission rings of the NIC. Again after having ran for a while the function moving the packets away from the kernel will be disrupted and

the software interrupt scheduled for execution again.

The actual transfer of a packet happens by calling the same function that is called in the case of an empty queue. If the driver is busy handling a packet for example from another CPU when the kernel tries to transfer the packet to its transmission ring, the driver will notify the kernel of it. When this happens the kernel will re-queue the packet and schedule the transmission side software interrupt. From the transmission rings it is the device driver's responsibility to finally actually transmit the packets on the wire. Figure 9 summarizes the packet transmission process to a simple flowchart.

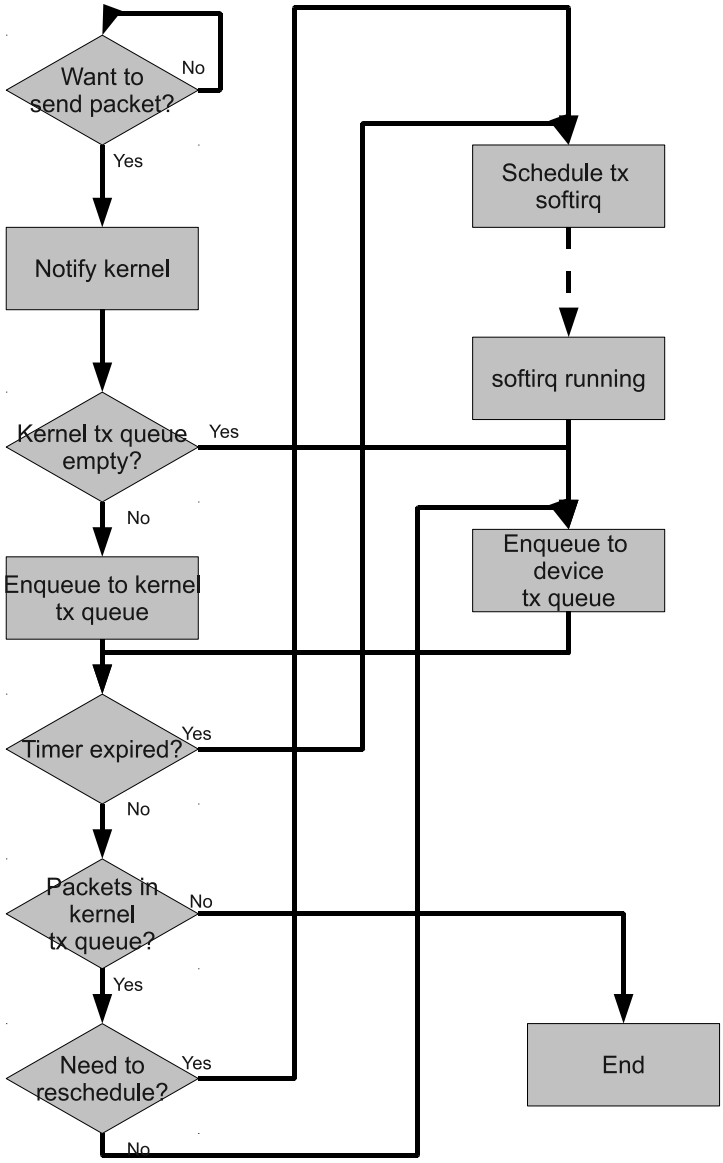

Figure 9: Basic flowchart of the packet transmission process.

To summarize, there are basically two different ways to transmit packets: first by transmitting them directly or second by going the software interrupt route. Obvi-

ously the first case is quite rare as there will at most times be several packets in the queue waiting for transmission, which leads to the software interrupt method being used. Figure 10 shows the "big picture" of the entities involved in packet reception and transmission.

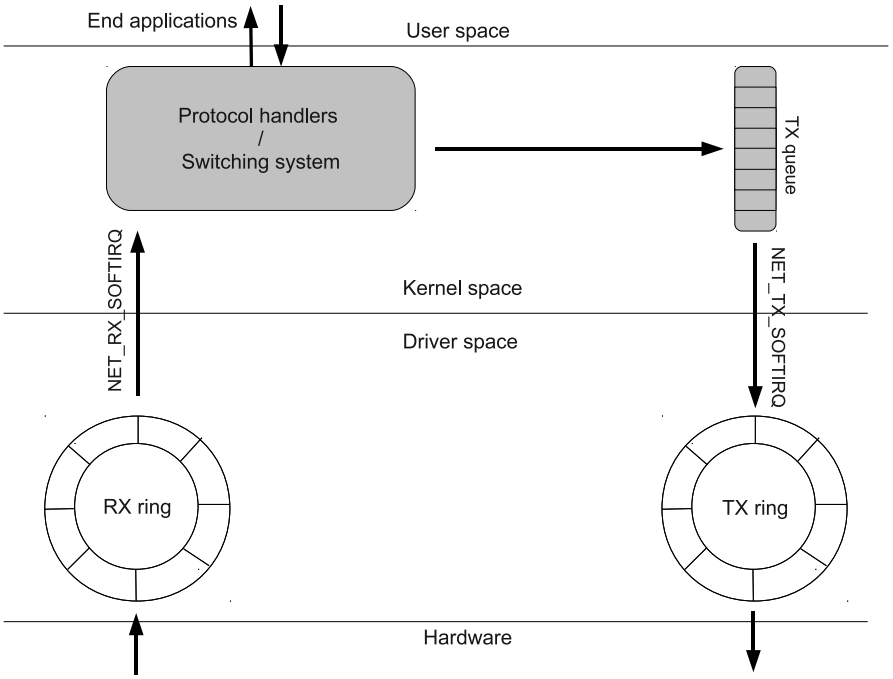

Figure 10: Big picture of the Linux network stack.

3.3 Layer 2 switching implementation

The actual switch device is implemented as a virtual device inside the Linux kernel. This virtual device is used to bind several physical interfaces together to form the switch and its ports. It also handles the forwarding of frames to the correct ports, and to do this it needs to keep track of the forwarding table. The decision that a packet should be handled by the switching system is made at the time when the packet is moved away from the NIC receive ring. When a packet is received and switching is enabled a special handler function is called for each packet that is received. This function does a fast check to see if the packet is actually meant for the switch or for some other protocol handler, after which it moves the packet to the correct handler. This check is done because the switch port can also receive other traffic that is not destined for the switch.

When it has been confirmed that the packet is actually destined for the switch, it checks if the packet is to be forwarded or if it is a Spanning Tree Protocol (STP) [45] packet that needs to be handled by its own handler. STP is a protocol that creates a loop free topology when several switches are connected together. It was not used during the tests executed for this thesis and will not be covered in more

detail. For both types of packets the forwarding database will then be updated with the packet's source MAC address and input port. In case there already is an entry for that particular address/port pair the the ageing timer will be reset to its original value. The timer is used to keep the forwarding database clean of entries that have not been used for a while.

After updating the forwarding database, the forwarding decision is made based upon a lookup in the forwarding database. If the destination address is not found or it is a multicast address, the packet will be sent on all but the input port. On the other hand if a match is found the packet will be sent directly to the correct output port. An interesting detail in the transmission side of the switch is that all the functions are called twice. This has been done in order to abstract the virtual device functions from the actual functions that do the packet sending on the physical device. So when a packet is sent from the switch, first all the transmission functions are called with the bridge device as an argument. When the final function is called, the statistics of the virtual device are updated and the process starts over but now with the actual physical device as an argument for the functions. This sequence of function calls finally leads to calling the function that queues the packet for transmission on the kernel side. From here on the process is identical to the one presented in Section 3.2.2. When the packet needs to be sent on all ports, a special function that goes through the switch's port list is called. For each port except for the input port the same procedure is repeated as with sending a packet on a single port.

3.4 NAPI

The old Linux 2.4 network stack was implemented in a way where each arriving packet made the network adapter generate an interrupt. In scenarios involving very high packet arrival rates, the system was so busy processing the interrupts generated by the network adapter, that no other program/task got any CPU time. This condition is called Receive Livelock [46]. It basically stops everything that is happening in a computer at that time. For a software router this essentially means that all packet processing and forwarding stops, rendering the router useless.

In order to evade the Receive Livelock problem NAPI [47] was invented. Before the actual NAPI implementation, some tests using an early dropping algorithm and the filling up of the backlog queue as a congestion indicator were done. These solutions were not perfect as they only worked with certain hardware that was able to slow down its interrupt rate. These designs formed the basis for the requirements considering NAPI.

The NAPI solution works by using a mixture of both polling and regular inter-

rupts. When the network adapter receives the first packet in a batch of packets, it sends and interrupt, which tells the system that the adapter is busy. Soon after the interrupt a software interrupt is scheduled. This small program starts polling the network adapter that for each round of polling sends a preconfigured amount of packets up to the network stack for further processing. When polling has finished, if the receive queue of the adapter is empty, interrupts will again be activated. On the other hand if the queue still has packets after the first polling round it will soon be polled again. If there are several network adapters present, each adapter that sends an interrupt will be added to a polling list. Each adapter on the list is then polled in a round robin fashion. Obviously if an adapter runs out of packets, it will be removed from the list and interrupts for it will be enabled again. If the adapter's receive queue gets full, all incoming packets will be dropped. In this way no CPU resources are wasted for packets that wouldn't get processed in any case.

Because the per processor backlog packet queues have been removed in NAPI, it doesn't suffer of packet reordering that happened in the early Linux kernels capable of multiprocessing. The combined polling and interrupt mechanism in addition to removing the Receive Livelock problem also gives a good balance between latency and throughput. When the packet arrival rate is low the system works in a similar fashion as a regular interrupt driven system offering low latency. On the other hand when huge amounts of packets arrive, the latency grows as packets have to wait longer for the poll to happen before processing of them starts.

3.5 Receive-side Scaling

In order to make better use of multicore processors in packet processing, a technology called Receive-side Scaling (RSS) [48] was invented. RSS is supported by network adapters that have multiple receive queues. By using multiple queues the packet processing workload can be spread, as each of these queues can be assigned to a certain core on the actual processor. Before RSS network adapters only had one receive queue that was assigned to one core. This meant that for example in a software router using a multicore processor, the processor resources were severely underutilized as only one processor core took care of the packet processing.

In RSS packets are multiplexed to the different queues using a Toeplitz hash function [49]. The hash value is typically calculated from the 4-tuple consisting of the source and destination IP-addresses, the TCP source and destination ports and a secret key. In case TCP would not be used, the hash would be calculated based on the 2-tuple consisting of the IP-addresses only. The function was selected so that the results produced either way would be distributed equally eavenly. Other things taken into consideration during the selection of the hash function were for example

ease of hardware implementation and randomness of the distribution. By using the 2- and 4-tuples for calculation it is guaranteed that the packets from the same flows are processed by the same processor. Thus no reordering of packets will happen inside of flows, and at the same time cache locality is preserved.

The Toeplitz hash function returns a 32 bit hash result. Out of these 32 bits only a preconfigurable amount between 1 and 7 bits is used in deciding which queue the packet should go to. The actual lookup is made in an indirection table that for each value of the masked hash result returns the right queue for the packet at hand. Figure 11 illustrates the steps taken during the RSS processing. All these features of RSS are implemented in hardware as software wouldn't be able to do these calculations at line rates of tens of gigabits per second.

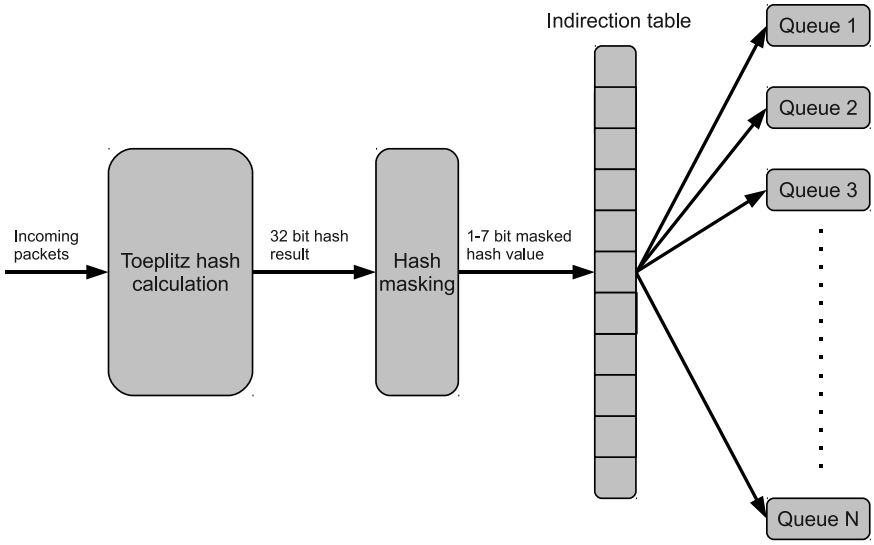

Figure 11: RSS processing sequence.

One can select to use as many queues as the hardware permits, but theoretically the best performance should be achieved using the same amount of queues per port as there are processor cores in the system. Using more queues than cores means that some cores have to handle the input of several queues, which might slow them down. Obviously if one does not want to allocate all processor resources to processing packets, one can use just a couple of the cores for packet processing and leave the rest for other tasks on the system. When building a software router it is still best to use all cores and the same amount of queues for packet processing.

3.6 IRQ affinity

By default all interrupts coming from the network adapters are handled by one processor core only. In cases of heavy load, this can lead to severe performance degradation, because the processor gets overloaded with interrupts. In order to spread the interrupts more evenly across all the processors, IRQ affinity [50] can be used. Affinity can be configured through the proc filesystem. Each interrupt has its own setting for affinity, that can be found in the directory /proc/irq/IRQ number/smp_affinity.

The smp_affinity variable is a bitmask that defines, which processor cores are allowed to handle the interrupt. The default bitmask in hex is 0xffffffff, which means that the interrupt can be handled by any processor core on the system. By setting the mask for example to 0x1, the interrupt will only be handled by processor 0. Similarly by setting the mask to 0x2 or 0x4 the interrupt would be handled by cores 1 or 2 respectively. It has been shown that by only using processor core affinity a 25% increase in throughput can be achieved [51]. Not only affinity for interrupts can be adjusted. Linux includes system calls for adjusting process affinity so that programmers can choose on which processor cores their programs are ran. This brings up the question of cache locality. Would it be beneficial to run the process that does the packet handling on the same core that handles the interrupts. According to some research conducted on this matter, process affinity also gives a small gain in throughput performance. All in all by using both interrupt and process affinity a 29% throughput gain can be achieved.[51]

3.7 Optimizing Linux and x86 performance

Section 2.8 presented some means of optimizing performance by making changes to the settings found in BIOS. The aim of this chapter is to present the software side of things, namely what kind of changes can be made to the OS, kernel and device driver to increase the performance of the system. Similarly to the hardware optimization case, one should first establish a baseline that can be used to verify the effects of changes made. After this, the changes should again be made one at a time.

3.7.1 Optimization on the subsystem and driver levels

If it is found out that the CPU is the bottleneck of the system there are some standard tricks that can be used to improve performance. First of all, all unnecessary programs should be shut down to free more system resources. Secondly one can modify the *priority of CPU intensive processes*, that are not critical to the system, to a lower value so that they are scheduled less often. Processes should also be bound

to one CPU so that cache flushes caused by processes hopping from CPU to another can be avoided. Also the interrupt load should be spread to several cores using *IRQ affinity*. If these measures don't help there is always the choice of upgrading to a faster processor in the case of single threaded applications or to increase the number of CPUs in the case of a multi threaded applications [37].

The most typical memory bottlenecks are related to how applications use memory and how the OS manages usage of the physical memory and page space on the hard drive. These can be optimized for example by adjusting page sizes and by adjusting handling of active and inactive memory. If it seems that the problem lies in that there is not enough memory available, one can try and disable unnecessary programs, processes and daemons to free up some memory or just simply upgrade to more memory.

Finally for tuning the network subsystem one can use both different kernel parameters and driver settings. Some of the simpler things one can do is, increasing the *Maximum Transmission Unit (MTU)* so that bigger frames get transmitted, increase sizes of *network buffers* or increase the length of the *transmit queue* on the kernel side. Other optimizations include the different *receive offloads* that strive to move several packets at a time from the NIC reception rings to the network stack. Unfortunately these have caused some problems with routing and switching, at least with the Intel NICs and ixgbe drivers. Also TCP and socket buffer sizes can be increased but these do not really impact the performance with a packet forwarding workload.

By modifying driver settings one can change the amount of *RSS receive queues* in use on the NIC . This should be set to equal the amount of processor cores that one wants to use for receiving packets. The interrupts of the queues should then be bound to their own cores. To reduce the amount of interrupts that the NIC creates one can use the *Interrupt Throttle Rate (ITR)* parameter, that controls how many interrupts per second the device can create. By increasing the value one can lower the latency with the cost of increased CPU time spent on processing interrupts. Lowering it , on the other hand, increases latency and lowers CPU usage. If the driver supports *NAPI*, it should be enabled as it increases network performance while decreasing the interrupt load. If NAPI is enabled one can choose to change the *dev_weight* and *netdev_budget* parameters. These correspond to the NAPI weight and budget parameters respectively, and control the amounts of packets transfered from the NIC to kernel per polling round. Also the length of the NIC *reception and transmission* rings can be changed. [52]

3.7.2 Kernel configuration

Compiling the Linux kernel from scratch makes it possible to edit different features of the kernel configuration. This enables the user to choose for example which drivers and other features will be enabled in the kernel. The changes may dramatically affect the performance of the system. At the same time the size of the kernel is reduced, which is important in smaller embedded systems.

One of the most simple optimizations for the kernel, when used for packet forwarding, is to disable the Linux packet filtering framework also known as *netfilter or iptables* [53]. It can be used to build a firewall of the Linux server but for packet forwarding it just slows things down as each packet that enters the system is analyzed by netfilter. To make the kernel optimal for the CPU that the system is being ran on, the *Processor family* setting can be changed. The *Tickless System (Dynamic Ticks)* [54] option lets the user choose whether the kernel uses dynamic timer interrupts or timer interrupts that are constantly generated. By enabling this option power consumption is reduced and the processor can stay in idle states longer. This is especially important in laptops and other portable devices. A closely related parameter is the *Timer frequency* that controls the frequency of the ticks and at the same time the resolution of the system timers. It has a direct impact on system responsiveness.

One major part of the kernel, that affects performance, is the different power control settings. These settings adjust the way that the kernel can control the power consumption of the system by adjusting for example the CPU clockrate. From a performance point of view the most relevant settings are the ones related to *Advanced Configuration and Power Interface (ACPI)* [55] and *CPU frequency scaling* [56]. ACPI is a specification that defines a platform independent interface for power management and monitoring. CPU frequency scaling on the other hand is the part of the Linux kernel that lets programs, the user or ACPI to adjust the CPU operating speed on the fly. By disabling these features one can be sure that the CPU is running at full speed all the time and that no application or the operating system can change the clockrate. This increases performance at the cost of power consumption: the user can be sure that the processor is running at full speed all the time and that no applications can suddenly change the speed.

Software routers and switches require large bandwidth on the memory bus and low latency for the memory access to be able to handle tens of millions of packets per second. These cannot really be optimized in any other way than by buying the lowest latency memory that is available. Through the kernel one can try and switch between one of the two memory allocators available: *slab* [57] or *slub* [58]. The slab system allows for kernel modules and drivers to preallocate a cache for objects that are frequently used by the program. When this allocation has been done and the

program needs one of these objects, it then just takes one of the already allocated objects instead of having to make a new memory allocation for each one of them. When the program releases the object it is marked as being free, after which it can be reused. The kernel itself also preallocates some caches that are shared by all programs. The slab allocator is know to use a lot of memory for the preallocated caches, which is why the slub allocator was created. It works similarly to the slab one but has significantly simplified the handling of the caches and thus should scale better for large NUMA systems. The slub allocator is now also chosen as the default setting.[59]

3.8 Summary

This section introduced the Linux kernel and its features relevant to packet reception, transmission and forwarding. The journey of a packet entering the system was covered from reception at the NIC, via software interrupt to the kernel and switching system and finally through the kernel side transmit queue and transmission side software interrupt back to the NIC and out to the wire. Relevant technologies that have helped to increase the throughput of the network stack, namely RSS, NAPI and interrupt affinity were also covered. Finally a couple of sections were dedicated to covering how the Linux OS can be tuned on the subsystem and driver levels and by configuring the kernel. Table 2 summarizes the OS, kernel and driver parameters that were presented in this section.

Table 2: Summary of OS, driver and kernel settings that were thought to have an effect on forwarding performance.

Parameter	**Effect**
Process priority	Uncritical processes will not be scheduled as often.
IRQ affinity	Spread the interrupt load to several processor cores.
MTU size	Protocol overhead reduced as more data transmitted per packet.
Kernel transmit queue length	Amount of dropped packets caused by queue overflow reduced.
Receive offloads	Offload some of the packet processing work to NIC.
RSS queue count	Spreads the packet processing load to several processor cores.
Interrupt Throttle Rate	Decreases interrupt load.
NAPI	Decreases interrupt load by using a mixture of polling and interrupts.
dev_weight	Changes weight of device in NAPI.
netdev_budget	Changes polling budget in NAPI.
NIC rx and tx ring length	Sets the length of the reception and transmission rings of the NIC.
Disable netfilter	Disables packet filtering framework that is not needed for forwarding.
Processor family	Optimizes the kernel for the processor that is used.
Tickless	Controls the method used for generating system ticks.
Timer frequency	Controls the frequency of the system ticks.
ACPI	Disables unnecessary power control features.
CPU frequency scaling	When disabled does not let the kernel control CPU clock frequency.
Memory allocator	Controls the way that the kernel allocates memory.

4 Test scenario, results and analysis

This chapter introduces the test scenario and setups used for the tests performed. First, a look at what the current Internet Engineering Task Force (IETF) standard defines about measuring device throughput is taken. After this the server setups will be covered and the actual execution of the tests described. Finally the results are presented and analysis of them done.

4.1 RFC 2544

Request For Comments (RFC) 2544 [60] defines the setup and results reporting styles for a number of different tests. Most of the tests are performed such that the output ports of a test generator are connected to a set of input ports on the Device Under Test (DUT). Figure 12 illustrates this basic setup. The output ports of the DUT are then connected to input ports on the same test generator. In this way it is relatively easy to measure things like throughput, frame loss and latency. RFC 1242 [61] defines all the terminology needed in the measurements. The most important term regarding the measurements performed for this thesis is throughput. Throughput is defined as

> "The maximum rate at which none of the offered frames are dropped by the device."

The throughput should be reported at different frame sizes ranging from the minimum to the maximum frame sizes available on the DUT. This gives the people interested in the test results a good overview of what the DUT is capable of. A loss of a single frame can cause big disturbances in the upper layer protocols and thus no frame loss is tolerated in the throughput test. Another of the more important measures is the frame loss rate that is defined as follows

> "Percentage of frames that should have been forwarded by a network device under steady state (constant) load that were not forwarded due to lack of resources."

The frame loss rate is mostly used to characterize the behavior of the DUT under overload conditions.

In addition to defining how the tests should be performed RFC 2544 also gives some pointers to how the results should be presented. The results of a throughput test should be reported in a graph were the y axis represents the frame rate and the x axis the frame size. The graph should also show the theoretical maximum frame rates for each frame size for comparison. If the performance of the DUT will be represented using one value only then the forwarding rate for the minimum

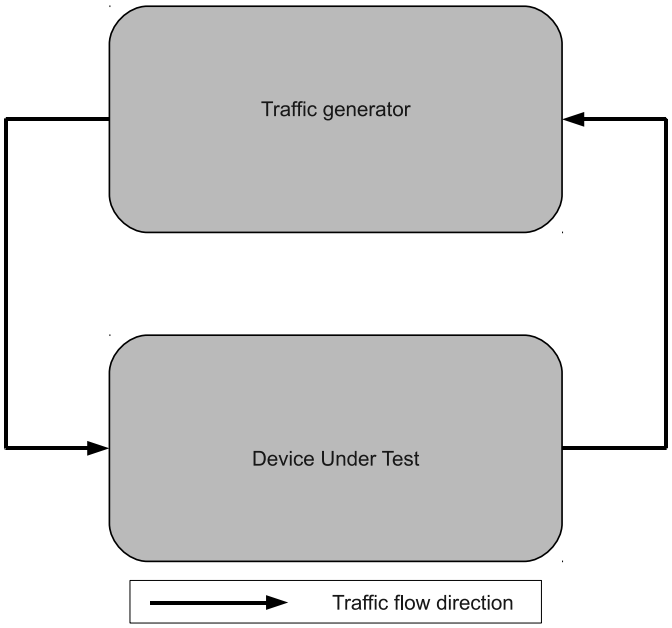

Figure 12: Basic test setup according to RFC 2544.

frame size should be used. The unit is to be frames per second. In addition also the measure in bits or bytes can be used. The frame loss rate should be reported similarly with the x-axis representing the percentage throughput of the maximum value and the y axis the percent loss at the particular frame size. The tests ran for this thesis were all performed according to the guidelines given in RFC 2544. Results reporting also follows these rules.

4.2 Server configurations

All the tests performed for this thesis were ran on two different servers. The first one was a regular single processor server consisting of a Supermicro X8STE motherboard [62] equipped with one six core Intel Xeon X5650 [63] processor running at 2667 Mhz. The server had 6 GB of of 1333 Mhz DDR3 memory. The second server was a NUMA machine with a Supermicro X8DTH-iF [64] motherboard equipped with two Intel Xeon X5650 processors and 12 GB of 1333 Mhz DDR3 memory. The memories were grouped such that each NUMA node got 6 GB of local memory. For both servers the memories were installed such that all three channels of the memory controller were utilized. The servers were equipped with two Intel Ethernet X520 Server Adapters [65], consisting of two 10 Gbps Ethernet ports each totaling to 4 ports per server. For the dual processor servers, the NICs were installed such that each NUMA node had one NIC. Table 3 lists the server hardware setups.

Both servers were running the Debian 6.0.1 “squeeze” OS using the Linux 2.6.32.41 kernel. The operating system was installed with a minimum installation comprised

Table 3: Setups of tested servers.

	Single processor server	Dual processor server
Motherboard	Supermicro X8STE	Supermicro X8DTH-iF
Processor	Intel Xeon X5650	2x Intel Xeon X5650
Memory	6 GB 1333 Mhz DDR3	12 GB 1333 Mhz DDR3
Harddrive	250 GB 7200 RPM SATA	
NIC	2x Intel X520 Server Adapter	

of only the command line interface and the necessary applications to get the server up and running. The choice of abandoning the kernel that was shipped with the OS distribution was made, as changing the settings and compiling the vanilla kernel is an easier task. The same mainline kernel version as the one shipped with the OS was used though. For the NICs the ixgbe-3.3.9 [66] driver was used. Most of the driver parameters were left at their default values for all the tests except for the tests considering them. For both servers the amount of RSS queues were set to the amount of logical cores, that is 12 and 24 queues for the single processor and dual processor machines respectively. The interrupts of the queues were spread evenly on all available processor cores. Large Receive Offload (LRO) and Generic Receive Offload (GRO) were switched off as they have been shown to cause issues in both bridging and routing [52]. Ethernet pause frames were also disabled for all tests, as pausing transmission during throughput tests would affect the results severely. No other changes to the default kernel, BIOS or driver settings were done after installation of the system except for the driver settings mentioned before. For each test, only the tested parameter was changed and all others kept unchanged at their default values.

4.3 Test execution

The layer two forwarding performance of both servers was tested with frame sizes of 64, 128, 256, 512, 1024, 1280 and 1518 bytes. The traffic generation was handled by a Spirent TestCenter [67] chassis equipped with two dual port 10 Gbps Ethernet cards of the Spirent TestCenter Hypermetrics CV model [68]. The traffic generator was connected to the servers according to the guidelines of RFC2544 so that the traffic would pass from the generator through the server and back to the generator. The traffic generator also does all the measurements in addition to traffic generation. The Linux kernel includes its own layer 2 forwarding implementation, which was used for the tests. Switching was configured between the 10 Gbps interfaces using the brctl utility.

The traffic generator was configured such that for each port 30 endpoints with

their own IP-addresses were configured. The endpoints are needed to get some variation to the protocol headers in the packets so that the RSS receive queues get utilized. Using only one endpoint per port would degrade the performance as only one of the RSS queues would be used. The endpoints were then paired so that all the endpoints of one port would send data to the 30 endpoints of another port and vice versa. Thus, at the maximum traffic rate, a traffic stream of 10 Gbps divided to 30 smaller streams is sent from each port. The traffic flows were routed such that they would pass from one NIC to another so that the server would be forced to do the maximum amount of work to forward the packets. For the single processor server this would mean that the packets travel from the input NIC through the PCIe bus to the processor and again back through the PCIe bus to the output NIC. On the dual processor server the packets would in addition to this have to travel from one NUMA node to another.

The maximal throughput result is found out by the traffic generator by using a kind of binary search algorithm. The tests start at 10% throughput. If the server is able to forward this amount of traffic then the next test will be ran with the value found at the middle point of the passed value and the maximum value. If the test fails the next value will be chosen from the middle point between the failed value and the previously passed value. In this way the test generator then iterates to find the actual maximum throughput for each packet size with 1% accuracy. For each throughput value the traffic is run continuously for 30 seconds. Each of the test cases that were chosen for a more thorough inspection were repeated three times to verify the results. As RFC 2544 states, the throughput is the rate at which no frames are lost by the device. Due to an unknown problem, both of the servers kept discarding packets in such amounts that using this assumption, they would have failed all of the tests. That is why the tests were run with an acceptable packet loss rate of 0,01%, as this gives a more realistic picture of the actual performance.

4.4 Calculating frame and data rates

Before the results are presented it is important to show what the theoretical maximum throughputs for the 10 Gigabit Ethernet are and how they are calculated. To calculate the theoretical maximum frame rates from this value, the 7 byte preamble, 1 byte Start Frame Delimiter (SFD) field and 12 byte inter-frame gap need to be taken into account in addition to the actual frame size [69]. The preamble is used to synchronize the receiver with the sender. It consists of 7 bytes of the type 10101010. The SFD tells the receiver that the frame starts. It contains 10101011 to differentiate it from the preamble. The interframe gap of 12 bytes allows the devices time to process the previous frame before the next one is transmitted. Equation 1 shows

how to calculate the frame rate when the mentioned factors are taken into account.

$$\text{Frame rate} = \frac{\text{MAC bit rate}}{(\text{Preamble} + \text{SFD} + \text{Frame Size} + \text{Interframe gap}) * 8\text{bits}} \quad (1)$$

The theoretical maximums for each frame size used in the tests for this thesis are calculated in Table 4. In addition the final column shows the theoretical maximum forwarding rate that can be achieved with four Ethernet ports per server.

Table 4: Theoretical maximum frame rates for 10 Gbps Ethernet.

Frame size (bytes)	**Theoretical maximum (fps)**	**Theoretical maximum forwarding rate (4 ports) (fps)**
64	14,880,952	59,523,809
128	8,445,945	33,783,783
256	4,528,985	18,115,942
512	2,349,624	9,398,496
1024	1,197,318	4,789,272
1280	961,538	3,846,153
1518	812,743	3,250,975

Finally, to calculate the the throughput in bits per second one needs the frame rate and size in addition to the preamble, SFD and interframe gap. The calculation is done according to Equation 2. Obviously the throughput at the theoretical maximum frame rate is always 10 Gbps for 10 Gigabit Ethernet.

$$\text{Throughput} = (\text{Frame size} + \text{Preamble} + \text{SFD} + \text{Inteframe gap}) * \text{Frame rate} * 8 \quad (2)$$

4.5 Single processor server results

This section will present the actual results of the measurements performed on the single processor server. The dual processor server results will be presented in Section 4.6. All the results presented in this section have been verified, by repeating the tests three times. For each frame size the maximum value of the three tests was chosen for the final result. The results that had a positive effect on the throughput performance are presented first. The results qualified as positive if a 5% increase was achieved compared to the baseline throughput.

Note that the left y-axis showing the throughput performance in frames per second has been cut off at 11 million for all the graphs. Leaving it at the actual maximum value of some 60 million frames per second, would have made the graphs

more or less unreadable, as the actual bars showing the results would have been very small. The other y-axis shows the throughput in Gbps. The maximum value for this is 40 and the whole axis is shown in the picture. All the graphs will show the actual results and both the theoretical maximum and baseline for comparison. The numerical results will be shown in Appendix A.

4.5.1 Baseline

The baseline tests have been ran with the all the kernel, BIOS and driver settings at their default values. These are the results that are used in later tests to see if the changes had any effect on the performance. Figure 13 shows the results. The results for frame sizes of 512 bytes and less, are quite modest. For the smallest two frame sizes not even 10% of the theoretical maximum throughput is achieved. At frame sizes of 1024 bytes and more, the throughput stays at some 60 to 70% of the theoretical maximum. An interesting observation is that for frame sizes ranging from 64 to 512 bytes the throughput in frames per second stays almost constant at 3 million, which for 64 and 128 byte frames equals to less than 10% of the theoretical maximum.

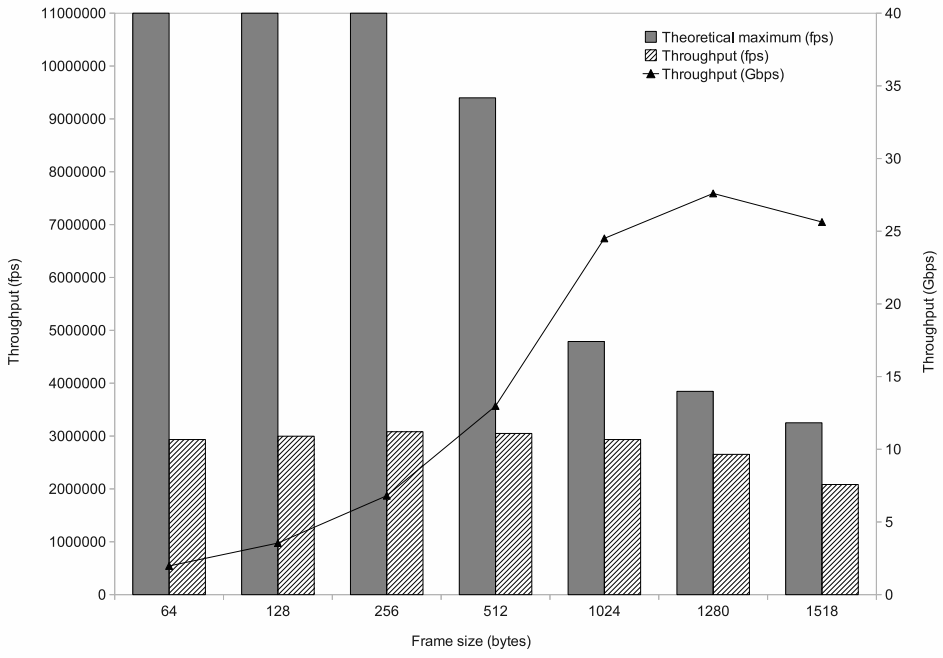

Figure 13: Baseline results.

4.5.2 Slab allocator enabled

Figure 14 shows the test results with the slab allocator enabled instead of slub. The effects are quite clear; the throughput of packets with sizes from 64 to 512 bytes has

increased by some 35% compared to the baseline. The throughput for these sizes is now around four million frames per second. The throughput for 1024 byte frames has increased by 11%, while the rest of the frame sizes show similar performance as the baseline. Just like the baseline case frame sizes of 512 bytes or less show a nearly constant throughput of around 4 million frames per second. Still the results are far from the theoretical maximums.

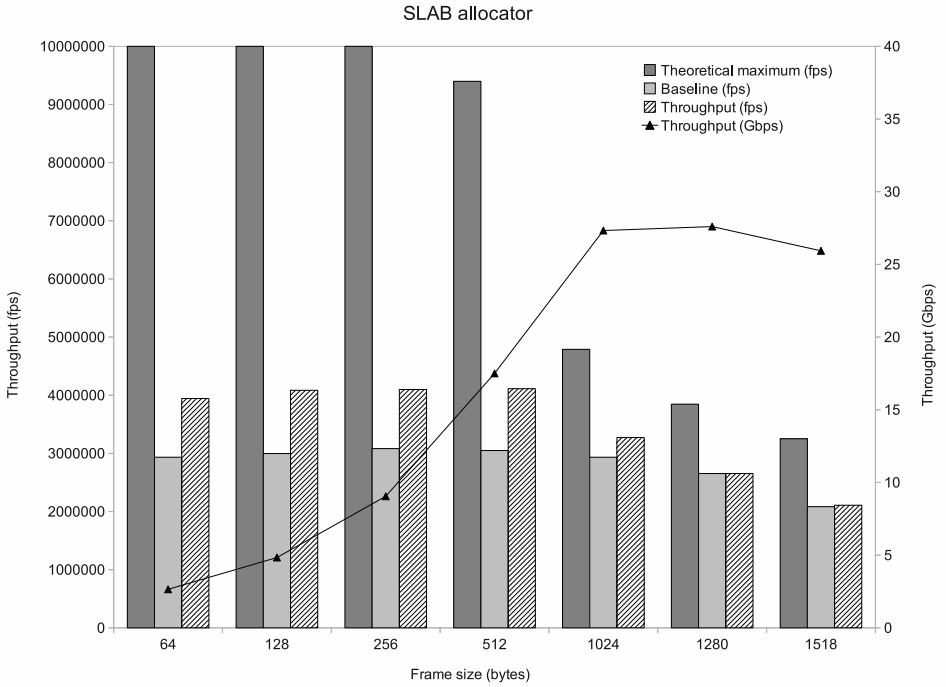

Figure 14: Results with slab allocator enabled instead of slub.

4.5.3 Netfilter disabled with slab allocator

By disabling the built in packet filtering framework netfilter alone, one does not get any increase in performance. Disabling it in conjunction with changing the memory allocator to slab on the other hand, gives a significant performance improvement. Figure 15 shows the results for this test case. For the three smallest frame sizes the performance has more than doubled and for 512 byte frames the performance is nearly doubled compared to the baseline. 1024 byte frames show a 7% increase and the rest are at the baseline performance except for 1280 byte frames that show a small decrease. The theoretical maximums are still far from achieved, but at least all frame sizes now achieve more than 10% of the maximum performance.

4.5.4 Disabling power management features

The common nominator for all the previous test cases presented was that they all mostly affected frames that were 1024 bytes or less in size. Changing power man-

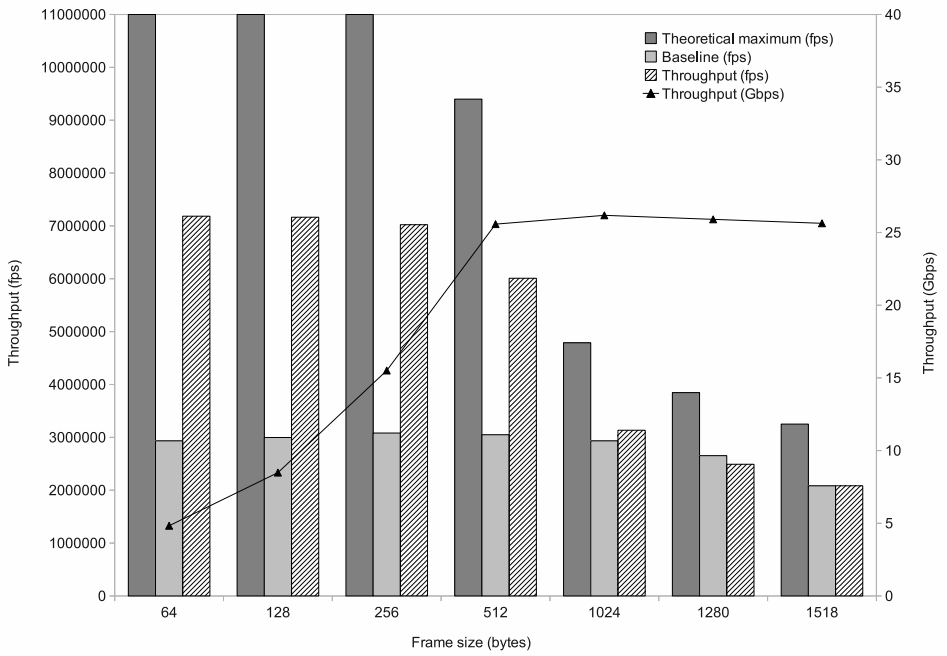

Figure 15: Results with netfilter disabled and slab allocator.

agement settings has a completely opposite effect. This test was run with most of the power management support except ACPI, Simple Firmware Interface (SFI) and CPU idle power management support disabled. In addition some of the ACPI specific modules were left out of the kernel namely AC adapter, Battery, Fan, Processor, PCI slot detection driver, container and module devices, memory hotplug and smart battery system. By doing these changes the throughput for frames of sizes 1280 and 1518 bytes increased by 14 and 44% respectively. This effect is shown in Figure 16. The throughput for other frame sizes stayed at the baseline level. 1280 and 1518 byte frames reached 79 and 92% of the theoretical maximum throughput. An interesting feature of this result seems to be that all frame sizes have a throughput of around 3 million frames per second. It was noted that by disabling the power management features the previously tested settings seemed to have a bigger effect also on frames of smaller sizes and thus these settings were left as they were for the rest of the tests. Subsequent test results will be compared to this one and not the original baseline.

4.5.5 Interrupt Throttle Rate set to 956

Several values for the ITR parameter were tested but only the minimum value of 956 seemed to have any effect. Figure 17 illustrates the results. For frames of sizes ranging from 64 to 1280 bytes the throughput increased with 21 to 28% and for 1518 byte frames the throughput is 8% higher and nearly reaches the line rate of 40

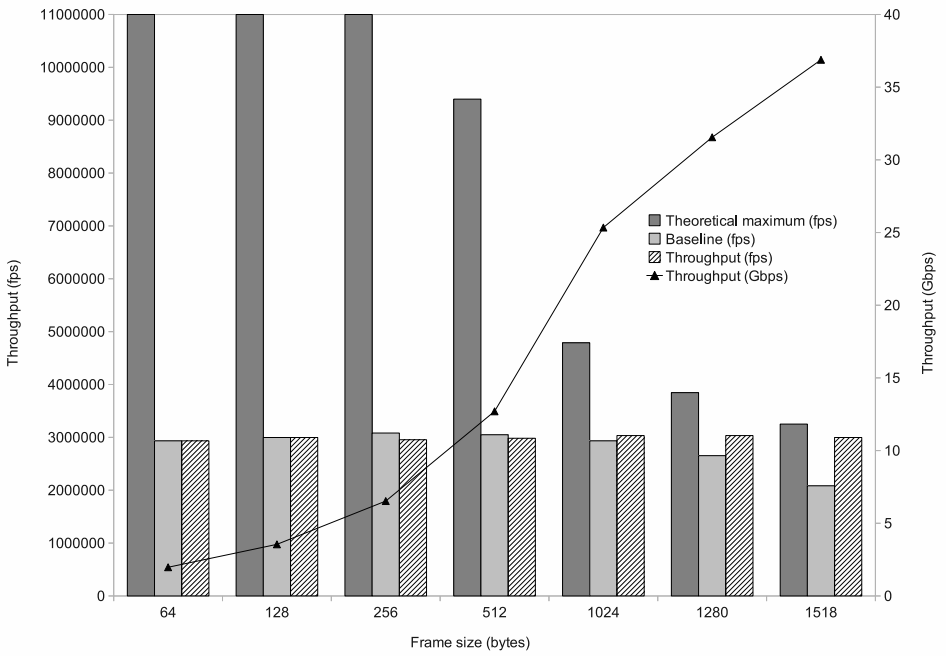

Figure 16: Results with power management features disabled.

Gbps. 1280 byte frames achieve 96% of the theoretical maximum, but the rest are still far away and reach only less than 80% of it.

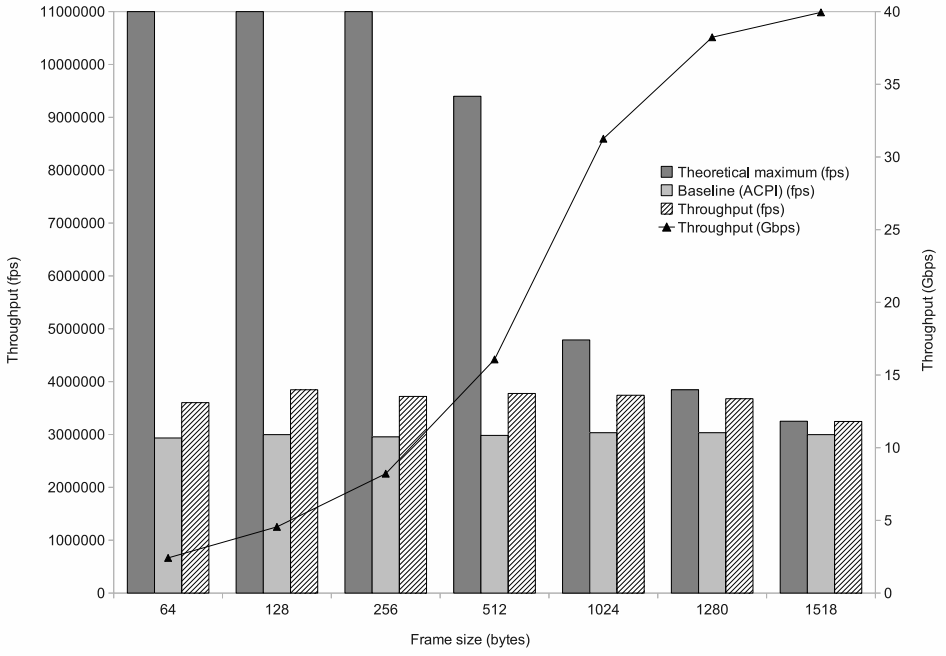

Figure 17: Results with Interrupt Throttle Rate set to 956.

4.5.6 Receive ring length set to 64

The default length of the NIC receive ring is 512 packets. By reducing the queue length to its minimum value of 64 packets one can gain in throughput performance. This value is also equal to the default value of the weight parameter of NAPI. Figure 18 shows that throughput of frames ranging in size from 64 to 1280 bytes, increased by roughly 20%, while the 1518 byte packets received an increase of 8%. As has been seen from previous cases the throughput of frames seems to stay at a constant rate regardless of the size. The two largest frame sizes seem to get close to the theoretical maximum throughput but as in Section 4.5.5, the rest of frame sizes are still far from the theoretical maximum.

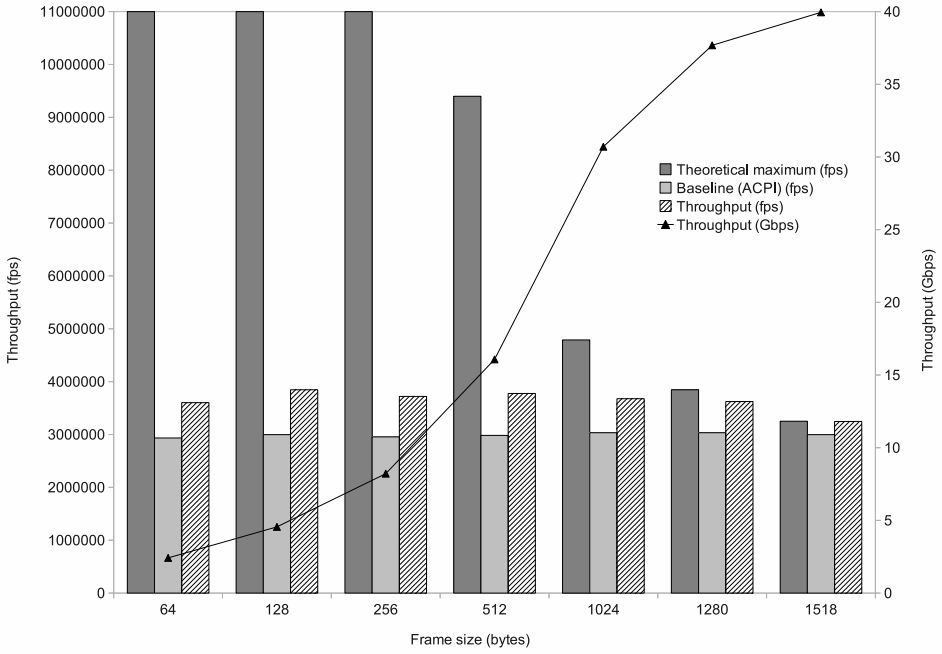

Figure 18: Results with NIC receive ring length set to 64.

4.5.7 Getting the most out of the system

As can be seen, some of the settings affect large frames while the others only affect the smaller ones. By combining all of the settings that had a positive effect, one could think would give the best performance. Unfortunately this is not the case. By combining all the settings that have been mentioned in the test cases presented in this section, the performance is good but not the best possible. The throughput of frames in the sizes of 64 to 256 bytes with this combination is around 6 million frames per second, which accounts to some 10 to 30% of the theoretical maximum. Looking at the two largest sized frames, the throughput is less than for example the

two previous test cases presented, which means that this is not the most optimal case.

The largest throughput for the single processor system is achieved by disabling the power control settings, enabling the slab allocator and disabling the netfilter packet filtering framework. Figure 19 shows the results for this test case. Yet again it seems that the smaller frame sizes experience nearly constant throughput. For frame sizes ranging from 64 to 512 bytes it equals to a throughput of some 7 million frames per second. Compared to the baseline this is an increase of 133 to 145%. The three biggest frame sizes get a forwarding rate of 95 to 100% percent of the theoretical maximum throughput.

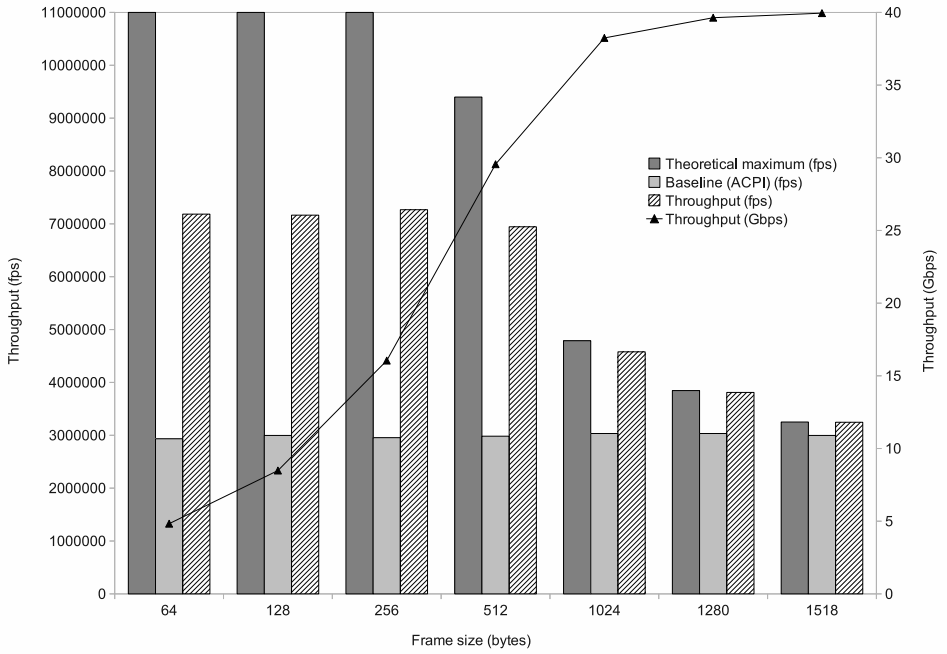

Figure 19: Results with power control disabled, slab allocator enabled and netfilter disabled.

4.6 Dual processor server results

This section presents the results of the tests performed on the dual processor server. The graphs are identical to the ones used for the previous results but the baseline is obviously changed to the one that was measured with this server. The numerical results will be presented in Appendix B.

4.6.1 Baseline

Figure 20 shows the results that were obtained with the dual processor server with the default settings. As can be seen the results are really quite poor. The through-

puts of frame sizes ranging from 64 to 256 bytes is around 4 million frames per second, but drops to half of that for the 512 byte frames and finally to less than a million frames per second for the rest of the sizes. These values correspond to throughputs ranging from 7% to 23% of the theoretical maximums. As with the single processor server there seems to be a consistency in that the smaller frame sizes receive constant throughput. The interesting thing here is that this server seems to perform worse than the single processor server with the same settings. This was not expected.

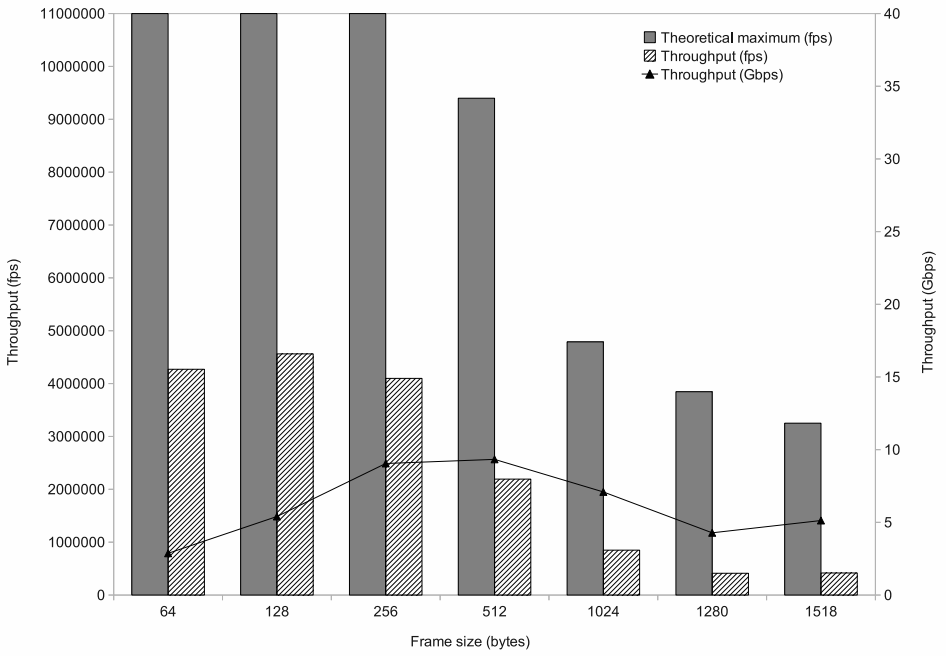

Figure 20: Dual processor baseline results.

4.6.2 Slab allocator enabled

Similarly to the single processor server, the dual processor server's forwarding performance increases significantly by enabling the slab allocator. Figure 21 illustrates these results. The throughputs of frames in sizes ranging from 64 to 256 bytes increased to nearly 5 million frames per second. At the same time the throughput of the 512 byte frames increased by nearly 40% and is at some 3 million frames per second. The rest of the sizes are still far below the 1 million frames per second mark. Interestingly the throughput of 1024 byte frames drops by 35% compared to the baseline. All the other frame sizes have increased their throughputs with 9% to 39%.

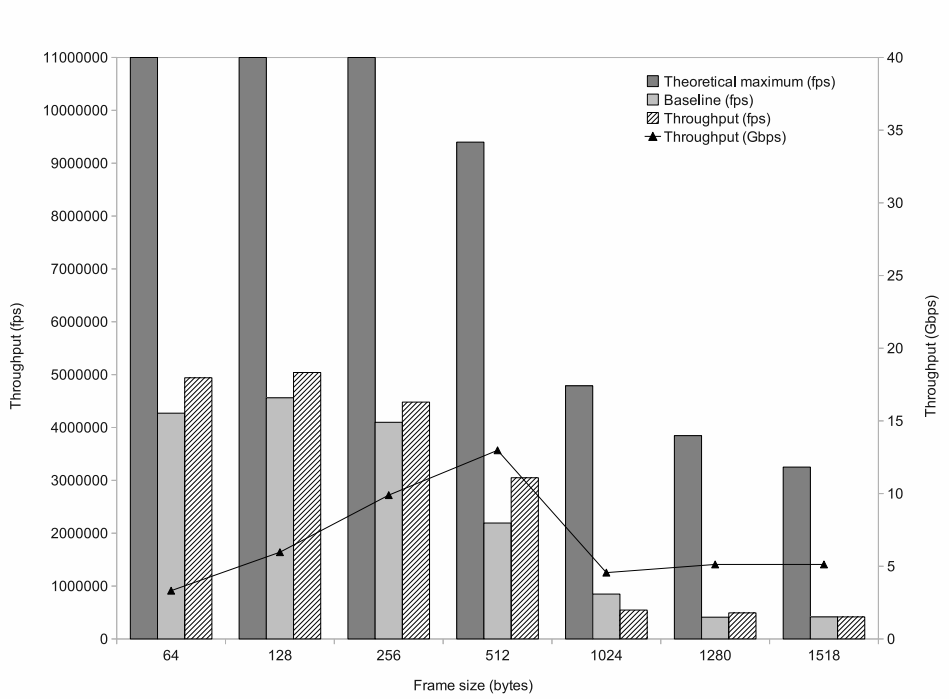

Figure 21: Dual processor server results with slab allocator enabled.

4.6.3 Netfilter disabled

Disabling the network packet filtering framework alone on the single processor server did not have any effect on the throughput. By doing this modification to the dual processor server, an increase in performance is achieved. Figure 22 shows that 64, 128, and 1280 byte frames throughput has increased by 15, 5, and 46% respectively. On the other hand all the other frame sizes experience a decrease in throughputs of 12 and 48% compared to the baseline. Theses values equal to 10 to 20% of the theoretical maximum throughput.

4.6.4 Tickless disabled

With the single processor server some parameters clearly affected throughput of large packets while others did the exact opposite. The same seems to be true for the dual processor server. By disabling the tickless kernel modifications that were created to save power, and thus letting the kernel tick away freely, throughput of frame sizes from 512 to 1518 bytes increased remarkably. This effect can be seen from the results in Figure 23. 64 and 128 byte frames see a throughput equal to the baseline result, while 256 byte frames see an increase of 9% in throughput. Starting from 512 byte frames the increase in throughput compared to the baseline is remarkable. It goes from twice as large for the 512 byte packets to 7.5 times as large with 1518 byte packets. Throughput seems to be constant at around 4.4 million frames per second for frames up to the size of 1024 bytes. An interesting observation is the

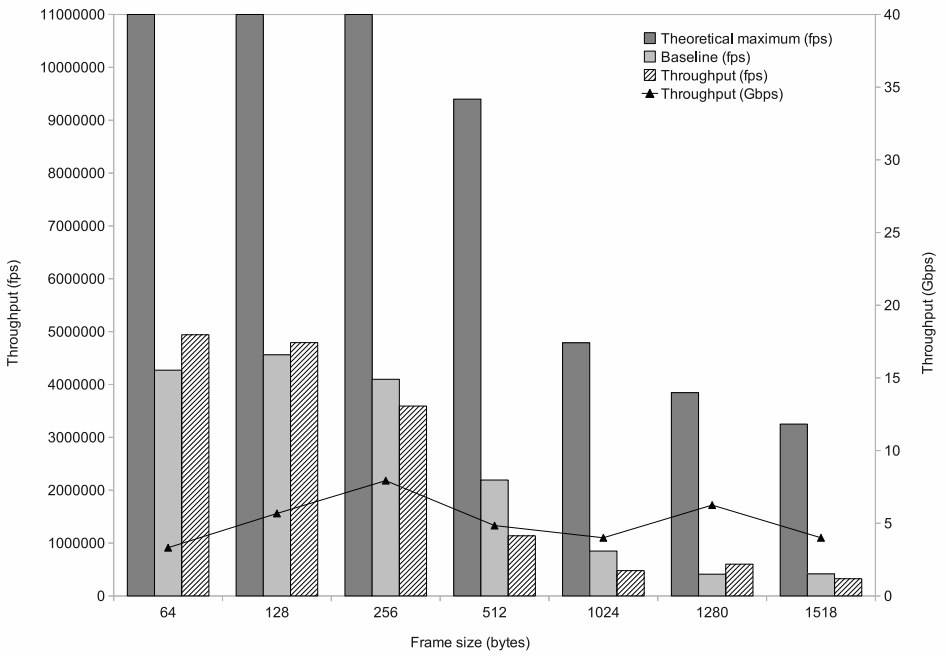

Figure 22: Dual processor server results with netfilter disabled.

reduction in throughput for 1280 byte frames. Although the increases are huge the dual processor still does not reach line rate for any frame sizes. The frame sizes that get closest to it are 1024 and 1518 byte frames that reach 91 and 96% of the theoretical maximum throughput respectively.

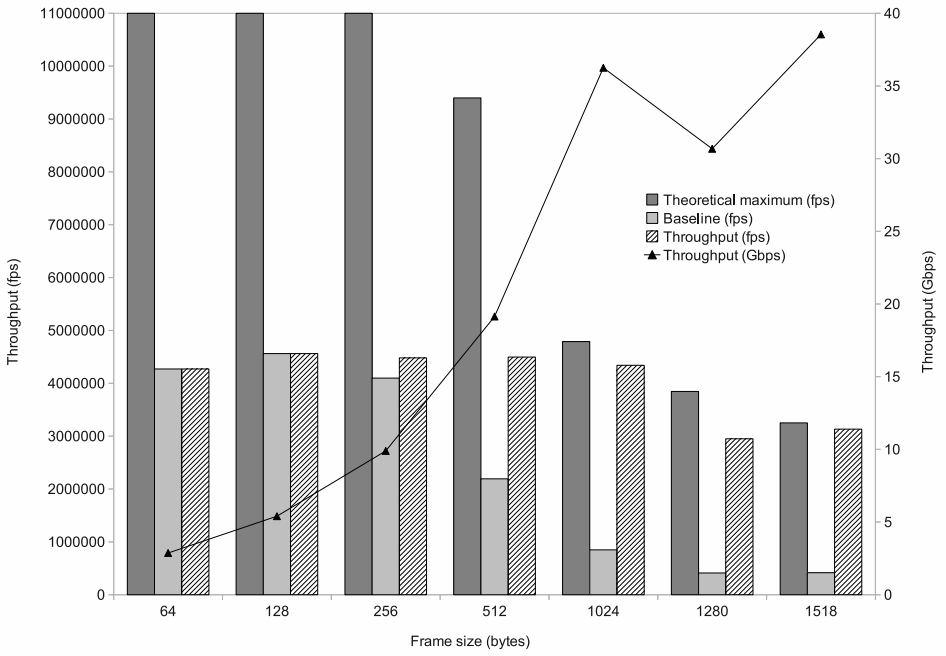

Figure 23: Dual processor server results with tickless disabled.

4.6.5 ACPI processor setting disabled

Another setting that only seems to affect the bigger end frames is the ACPI processor option in the kernel configuration. This test case is similar to disabling most of the power control features in the single processor server but a lot simpler as the same effect is reached by disabling only one setting. The results of this test are shown in Figure 24. Frame sizes ranging from 64 to 1024 bytes all seem to get a throughput that is constant at around 4.4 million frames per second. These throughputs equal between 8 and 92% of the theoretical maximums. The 1024 and 1518 byte frames are about one percentage unit from reaching the theoretical maximum throughput. Comparing the results to the baselines it is interesting to see that the throughput of 128 byte frames has decreased. All the other frames see a substantial growth. For example 1280 byte frame throughput is more than nine times larger than the baseline. Similarly to the single processor server, the ACPI processor settings was left disabled for the rest of the tests.

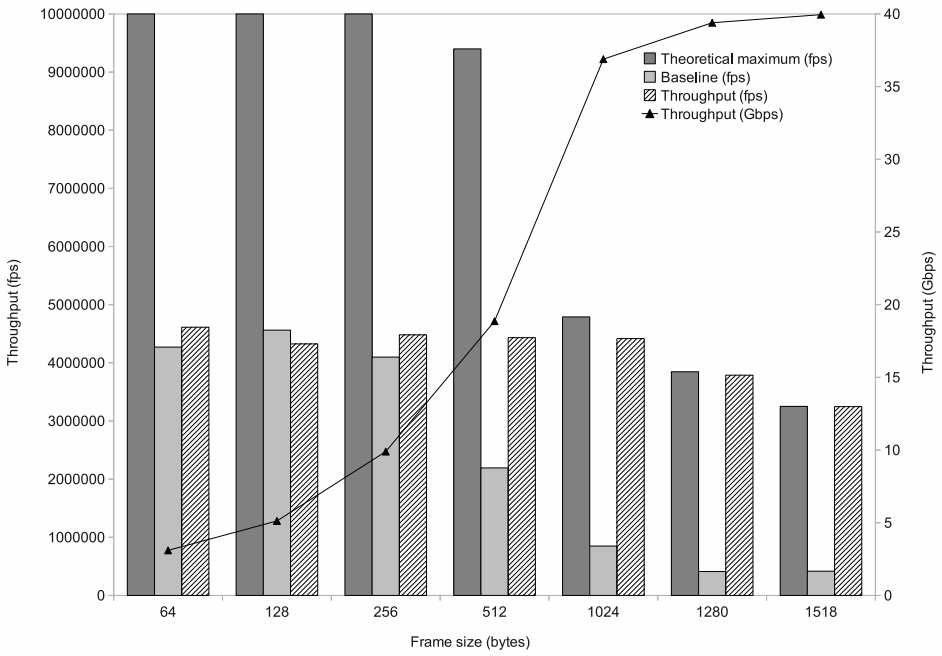

Figure 24: Dual processor server results with ACPI processor setting disabled.

4.6.6 Receive ring length set to 64

Changing the receive ring length of the NIC seems to have the largest effect on the smaller frame sizes. These results are shown in Figure 25. The increase in throughput is in the 10 to 15% range for frame sizes ranging from 64 to 512 bytes. 1024 byte frames see a 9% increase in throughput while 1280 and 1518 byte frames

are at the same values as the baseline. The throughput values seem to be around the 5 million frames per second mark for frames up to 1024 bytes of size. The 1024, 1280 and 1518 byte frames are all nearly at the theoretical maximum throughput but the rest of the frame sizes are still far away and reach only between 8 and 54% of it.

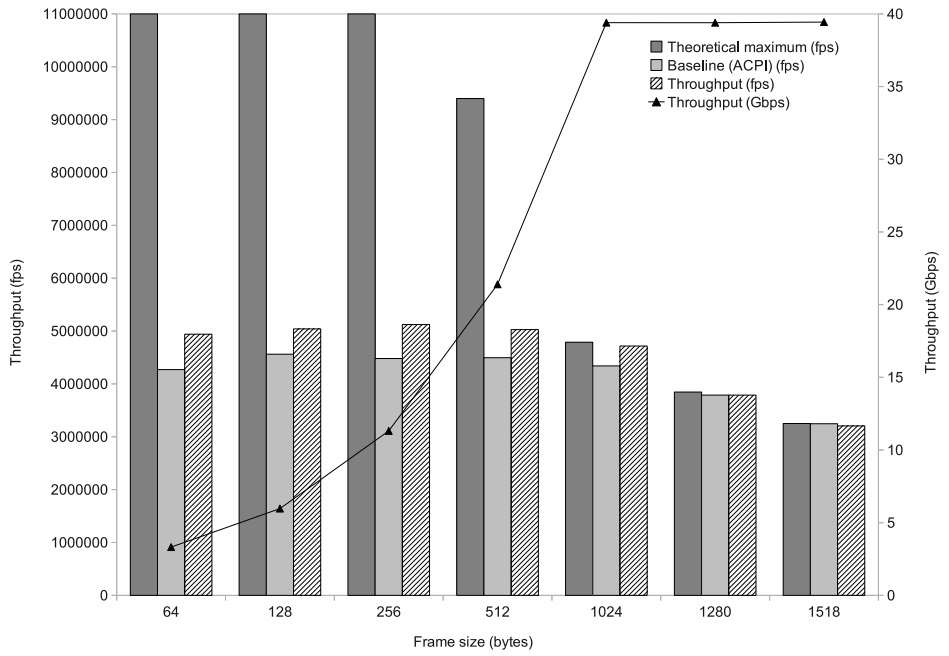

Figure 25: Dual processor server results with receive ring length set to 64.

4.6.7 Maximum performance

For the dual processor server it seems to be true that doing all the previous changes at one time gives the best throughput result in contrary to the single processor server, where only a subset of the changes gave the best result. Thus the results presented in Figure 26 have been obtained by disabling the ACPI processor setting, enabling the slab allocator, disabling netfilter, setting the receive ring length to 64 and disabling tickless. The first thing that should be noted is that this is the first test case where the throughput is more than 10 million frames per second. This is the case for 64, 128 and 256 byte frames. These frame sizes have also all doubled their performance compared to the baseline. 512 byte frames are forwarded at some 8 million frames per second which accounts for an increase of 78% compared to the baseline. 1024 byte frame throughput increased by 5% to 95% of the theoretical maximum and 1280 and 1518 byte frames are at the baseline values a couple of percentage units below the theoretical maximum.

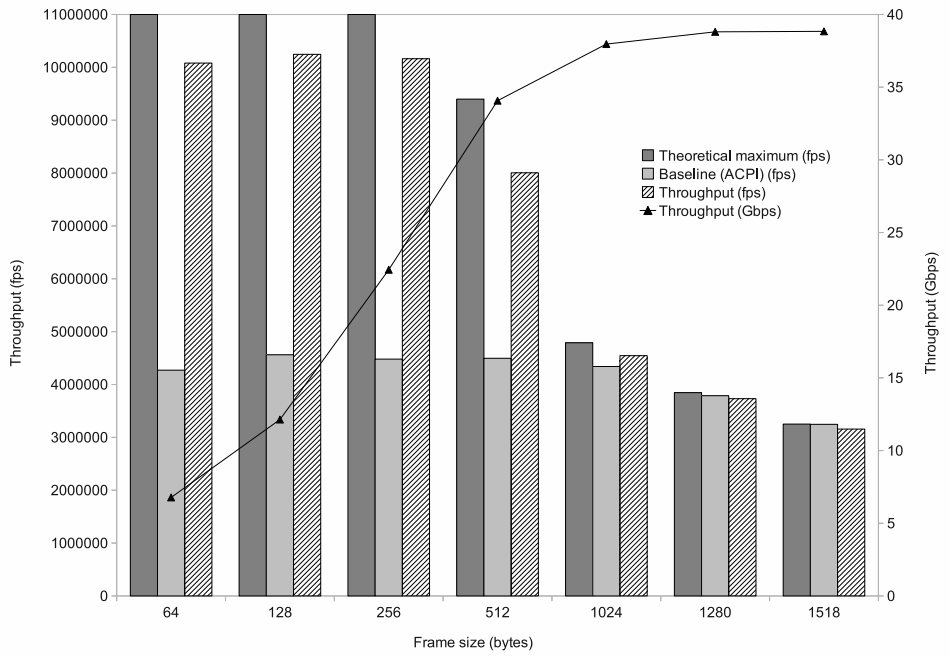

Figure 26: Dual processor server results with ACPI processor setting disabled, slab allocator, netfilter disabled, tickless disabled and receive ring length set to 64.

4.7 Settings that did not have an effect

The previous sections only show the results that had a positive effect on the throughput in bridging. Several other settings were also tested. Most of them had no effect at all, while for others the positive effects were so small that they could be counted as statistical variation, and thus they were not counted. It also seemed that most settings were already at their most optimal values and thus changing them only decreased the forwarding performance. In general the BIOS settings did not seem to have any effect on the performance. Obviously dropping the CPU clock multiplier or reducing the memory or QPI bus speeds had a negative effect on the performance. Changing the PCIe payload size or the different virtualization related settings (I/OAT, EIST and VT-d) did not have any effect on performance. The same applies for the memory channel and bank interleave, and prefetching settings as well as for the different power saving state related settings.

As for the kernel configuration, the timer frequency setting did not have an effect, although the tickless setting that is also related to the timers had a major effect on the dual processor server. Disabling the high resolution timer did not affect performance. Setting the processor type setting to match the type of processor that was installed, instead of the generic x86 setting, did not make any difference for the throughput. Changing the kernel preemption model to the no forced preemption setting had no impact. The driver settings were also more or less at their optimal

values. Setting the ITR to different values did not have such a large effect on the dual processor server as it had on the single processor one. Obviously disabling NAPI and reducing the amount of RSS queues did have a negative effect on performance. The drop was not as big as one would have imagined though. Finally the NAPI budget and weight parameters did not have a statistically significant effect on throughput.

4.8 Analysis

The aim of this section is to shed some light on how the single and dual processor servers performed compared to each other and on the other hand to try and identify some of the bottlenecks that limit their performance. First lets see how the servers compared with each other. The assumption is that the dual processor server should perform better than the single processor server. The dual processor server basically consists of two identical copies of the single processor server, that are attached together by the means of QPI. This could lead to making an assumption that the dual processor machine would forward frames twice as fast as the single processor one.

Table 5 shows the performance difference between the two servers for selected test cases. The numbers show the ratio of the dual processor server throughput to the single processor server throughput. Note that the ratios presented in italics cannot be larger than 1, as both servers are operating close to line rate throughput. As can be seen the dual processor server forwards packets maximally 57% faster than the single processor one. The percentage varies between 9 and 57 and typically seems to average around 35 to 40%. What is interesting is that for the baseline test and packets larger than 512 bytes in size, the throughput of the dual processor server is significantly lower than for the single processor configuration. Why the server behaves like this is still a mystery. Obviously for the bigger packet sizes the differences between the servers are small as both servers were able to forward packets of these sizes at or near line rates. A reason for the dual processor server not doubling performance compared to the single processor one, could for example be that the Linux networking stack has not been optimized for NUMA systems that are acting as switches and routers.

4.8.1 Finding bottlenecks

Finding the bottlenecks of the system was done by reducing the CPU clock frequency, the memory bus speed and amount of memory channels. By changing one of these values at a time, running the tests and checking for effects on the results, the bottlenecks of the system could be identified. If reducing one of the values did not have an effect on performance, it could be clearly stated that the particular

Table 5: Ratios of dual and single processor server performance.

Frame size (bytes)	**Baseline**	**Slab allocator**	**ACPI / Power management**	**Maximum performance**
64	1.46	1.25	1.57	1.4
128	1.52	1.23	1.44	1.43
256	1.33	1.09	1.52	1.4
512	0.72	0.74	1.49	1.15
1024	0.29	0.17	1.46	*0.99*
1280	0.16	0.19	1.25	*0.98*
1518	0.2	0.2	1.08	*0.97*

component was not a bottleneck of the system. On the other hand if the performance of the system was degraded, by doing the changes, it was concluded that that component was a bottleneck. The seriousness of the bottleneck was evaluated by comparing for example the ratio of the reduced bus speed and original speed to the ratio of the throughput at the reduced and original bus speed. If these two ratios would match, it was deemed that the throughput was directly limited by that component.

First the CPU clock multiplier was dropped from its maximum value of 20 to its minimum value of 12. The ratio of these values is 0.6. For the single processor server the effects were more pronounced than for the dual processor one. For all the test cases that were run with the 60% processor clock the frame throughput for the single processor server fell to around 65% of the original values for frames of sizes 64 to 512 bytes. As the frame sizes grew, the effects were more subtle. The server then managed to forward some 70 to 80% of the original throughput. This difference is explained by the fact the the bigger the frames get, the longer the arrival interval of them is, and thus the server has more time to process each frame. Similar effects could be seen for the dual processor server as the throughput of small packets fell to some 70 to 75% compared to the original values. Similarly to the single processor server the throughput of big packets was larger, and reached some 80 to 85% of the original. The conclusion is that for both servers the forwarding performance is limited by the processor speed, at least for the smaller packet sizes. Obviously with the dual processor server having double the amount of processing power available, the drops are not as large as with the single processor server.

Dropping the memory bus bandwidth to 800 Mhz did not have as big an effect as reducing the CPU clock. The ratio of the new bandwidth to old was again 0.6. For the single processor server, the throughput for smaller frame sizes was between the original value and 85% of it. The larger sizes on the other hand got some 80% of the original value. The reason for this obviously is that, the larger frames

have larger throughput in Gbps, which means that a larger amount of data passes through the server and memory bus, and thus for these frames the throughput is reduced, when the memory bus bandwidth is reduced. For the dual processor server the memory bus bandwidth does not seem to affect the results. This is probably because both the NUMA nodes have their own memory buses, which means that there is double the capacity for the same amount of data, compared to the single processor server. As it seems, the memory bus bandwidth did not severely limit the forwarding performance. This result is also clear when looking at the theoretical values of the memory bus bandwidth. By doing 800 million 64 bit transfers per second, the memory bus can transfer 51,2 Gbps. By multiplying this with the amount channels (3), one gets 153,6 Gbps as the theoretical maximum speed of the memory bus operating at 800 Mhz.

The amount of memory channels that were used was changed by removing some of the memory modules so that only one of the channels per memory controller was active. This meant that a ratio of 0.33 would mean that the memory channel would be a bottleneck for forwarding performance. Using the single processor server the smaller sized frames got between 50 and 80% of the original throughput, depending on the test case. The bigger sized frames throughput dropped to around 40%. Clearly the amount of memory channels is a bottleneck for the single processor server. For the dual processor server the effects were not as clear. Most frame sizes seemed to achieve a throughput of 80 to 90% compared to the original for all the test cases. Again, as with the reduction of memory bandwidth, the dual processor server has an advantage in that it has one memory channel active for each NUMA node. This is the reason to the reduction of channels not having such a large effect on it as on the single processor server. Thus, the amount of memory channels seems to be a bottleneck for the single processor server.

4.8.2 Further analysis

This section aims to shed some light on why the different settings had an effect on performance. These thoughts are entirely the author's and are not based on any experiments or facts. The netfilter packet filtering framework case should be quite clear at least. When netfilter is enabled all the packets that enter the system go through it even though no filtering rules have been enabled. With small packets at high rates this causes an unnecessary large amount of processing power being wasted by the framework and thus packet forwarding is rather slow. The slab allocator case is still a small mystery and would require more investigation but there has been some indications [70] that the slab allocator might be faster for some workloads compared to slub.

The ITR setting had an effect on the single processor server. This might be

because the processor has more time to process packets, when it isn't interrupted at such a high rate. Also the driver readme file states that the dynamic ITR mode, that is enabled by default, might increase CPU usage. Setting the receive ring length to 64 also increased performance. It might be related to the NAPI polling scheme. When the queue is shorter the time spent processing the software interrupt is shorter, which would free the processor for other tasks. On the other hand because of the shorter queue the software interrupt has to be scheduled more often, which adds its own overhead to the packet processing.

The power management, ACPI and tickless settings are all related in a way. They have all been invented to reduce power consumption by different means. Tickless removes the periodical timer interrupt that is generated 1000 times per second while the power management settings monitor the CPU usages and try their best to reduce clock speeds or shut down cores to reduce power consumption. These settings all had an effect on the bigger frame sizes, which could be because the packets arrive at so long intervals that the CPU has time to go to some sleep state. When the next packet arrives there is a latency to the processor waking up from the state and this then messes up the packet processing somehow. By disabling the power management and ACPI features one can be sure that the CPU is running at its maximum speed at all times. The effects of the tickless setting could be explained by the fact that the timer tick keeps the cores in the active state and thus the bigger packets get processed without the CPU going to sleep.

Finally, several test cases showed that the throughput in frames per second was constant for the smallest frame sizes. This would indicate that the bandwidth of the internal buses is big enough to handle the amount of data. On the other hand it also tells that there is some latency in the packet processing that is independent of the packet size. This could be either related to the memory access latency or to the time it takes for the switching program code to execute the instructions needed to forward the packet.

4.9 Summary

This section covered everything from test execution to the actual results and analysis. The chapter started with a look on what the relevant standards documents have to say about throughput testing. After this the tested servers, traffic generation equipment and their configurations were introduced. The results were then presented. The single processor server maximally forwarded some 7.2 million frames per second while the dual processor server reached a throughput of around 10 million frames per second. Finally some analysis on the results was conducted. It was concluded that for the single processor server the bottlenecks were the CPU and

memory channel amount. For the dual processor server there were not as many distinct bottlenecks. Reducing the CPU speed, memory bus bandwidth and memory channel count did not have as big an effect on the dual processor server as on the single processor server. This is obviously because the dual processor server is basically two of the single processor servers connected together. Still its throughput was not doubled compared to the single processor server, which would indicate that there is some overhead in the Linux network stack and NIC driver.

5 Conclusion

The aim of this thesis was to find out the performance of the Linux operating system when working as a software switch. Furthermore the performance was optimized and some bottlenecks identified. Software routers and switches have huge potential to become a solution that is used when a cost effective method for packet forwarding is needed. What they do not offer is the huge performance that is required in operator backbone networks. For these situations equipment that is constructed using ASICs and NPUs is required. These components offer less flexibility in creating new protocols and other related functions that might be needed in a router. This is where the software routers have their biggest advantage compared to regular network equipment. Because everything that the software router does with packets passing through it is done in software, new protocols and other required changes can be easily made by altering the source code. Most software routers run on some form of Linux operating system, which further increases the ease of making changes as anyone can access the code freely.

The typical software router is based on the x86 architecture. It has a motherboard that is used to connect the CPU/CPUs, NICs and other components together. The NICs are connected to the northbridge chip using the PCIe bus. The northbridge connects to the CPU by using the QPI bus, in the case of an Intel Nehalem processor. These buses in conjunction with the memory bus and processor form the hardware bottlenecks that limit the performance of a software router. The other key component in addition to the hardware is obviously the software that the router runs on. As was stated earlier the operating system of choice for software routers is Linux, because it includes implementations of both routing and switching. The journey of a packet through a Linux software router starts at one of the RSS receive queues from where it is transported, via software interrupt and NAPI up the network stack and into the switching function that decides which port the packet should be output to. From here on the packet then goes to an output queue from where it is transported by software interrupt to the NIC and onto the wire.

To optimize the performance of a server in general one needs to first establish a baseline that is used to see if the made changes had any effect on performance. After this the changes and optimizations can be made. The hardware settings can be changed in BIOS or by changing the driver parameters. The Linux operating system on the other hand can be optimized by changing the kernel configuration. For this thesis two servers, one with a single CPU and another with dual CPUs, were tested using the Linux switching implementation. Both servers had four 10 Gigabit Ethernet ports, which made it possible to compare the results. For both servers the performance was quite poor with the default settings of the distribution.

The biggest performance gains in small packets throughput for both servers were achieved by changing the slab cache allocator from slub to slab and by disabling the network packet filtering framework. Larger sized packets seemed to benefit more from disabling different settings related to ACPI and other power management features. By doing these changes the single processor server achieved a maximum throughput of some 7.2 million frames per second compared to 10 million frames per second on the dual processor server. These results were obtained with 64 byte frames. For the frame size of 1518 bytes both servers achieved the theoretical maximum throughput of around 3.25 million frames per second.

On could have assumed that the dual processor server would double the performance compared to the single processor one. This was no the case as the dual processor server only performed around 50% better compared to the single processor server. The bottlenecks of both servers were found to be the CPU processing power and memory channel count. The memory channel bandwidth did not have a large effect on the packet throughput.

There is a large potential in software routers to become work horses in the networks. Currently most small routers used at home do all the packet forwarding using software, while on the other hand big enterprise and operator networks still run on hardware based on ASICs and NPUs. To make the software routers more attractive some deficits need to be fixed though. First of all, using Linux, no changes to the interface configurations can be made without the interface resetting itself. In business critical applications this is a big downside, as traffic should not be disrupted. Secondly, the kernel code is not obviously optimized for forwarding packets. Significant effort has been put into the packet forwarding code by researchers around the world. Unfortunately everybody seems to be working only on their own things [5, 7, 71, 72]. By combining all the work done by researchers into one kernel or Linux distribution it would be a lot easier to get the software out to the end users. This would surely also increase the visibility of software routers to the general public. Finally, more vendors should offer software router products. As far as I know, at the moment only Vyatta is offering software routers for sale. By tackling these problems the generally negative attitudes to software routers could slowly start to change.

Future research topics in the fields of software routers could include for example the following things:

- Test performance on newer Intel microarchitectures (Sandy Bridge and Ivy Bridge): To see what the effects of changing microarchitecture are different architectures should be tested.

- Test performance using AMD microarchitectures (Bulldozer): Typically Intel CPUs have been considered to perform better than AMD CPUs, which is why

it would be interesting to see how they compare against each other under a packet forwarding workload.

- Test IP routing performance: IP routing performance is equally interesting as switching performance and should also be evaluated.
- Test FreeBSD performance: To see how another Unix based operating system performs in packet forwarding compared to Linux.
- Test performance using a newer kernel: The kernel version used for these tests dates back to 2009. At the time of writing this, the newest stable kernel version is 3.3.2. It would be interesting to see if the performance would increase by using a newer kernel.
- Further optimize the Linux code for packet forwarding: The kernel is obviously not optimized for a packet forwarding workload. Effort should be made to implement required changes in the kernel to reduce the packet processing overhead.
- Enable on the fly configuration: Software routers based on Linux will not be used in any business critical services for as long as changes to interface configuration require a reset of the interface.
- Detailed analysis of performance bottlenecks: The bottlenecks should be analyzed in more detail using for example profiling software to see how the network stack code behaves.

References

[1] "Cisco Systems Inc." `http://www.cisco.com`; accessed July 21, 2011.

[2] "Juniper Networks Inc." `http://www.juniper.net`, accessed July 21, 2011.

[3] "Vyatta Inc." `http://www.vyatta.com`, accessed July 21, 2011.

[4] R. Olsson, H. Wassen, and E. Pedersen, "Open Source Routing in High-Speed Production Use," in *Proceedings of Linux Kongress 2008*, October 2008.

[5] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader: a GPU-Accelerated Software Router," in *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, (New York, NY, USA), pp. 195–206, ACM, 2010.

[6] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek, "The Click modular router," in *Proceedings of the seventeenth ACM symposium on Operating systems principles*, SOSP '99, (New York, NY, USA), pp. 217–231, ACM, 1999.

[7] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "RouteBricks: Exploiting Parallelism To Scale Software Routers," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, (New York, NY, USA), pp. 15–28, ACM, 2009.

[8] J. F. Kurose and K. W. Ross, *Computer networking: a top-down approach*. New York: Pearson, 5th, international ed., cop. 2010.

[9] H. J. Chao, C. H. Lam, and E. Oki, *Broadband packet switching technologies: a practical guide to ATM switches and IP routers*. New York: Wiley, 2002.

[10] P. C. Lekkas, *Network Processors*. New York, NY, USA: McGraw-Hill Professional Publishing, July 2003.

[11] R. Giladi, *Network processors : architecture, programming, and implementation*. The Morgan Kaufmann series in systems on silicon, Amsterdam: Morgan Kaufmann, cop. 2008.

[12] N. Shah and K. Keutzer, "Network Processors: Origin of Species," in *Proceedings of ISCIS XVII, The Seventeenth International Symposium on Computer and Information Sciences*, 2002.

[13] G. Lawton, "Will network processor units live up to their promise?," *Computer*, vol. 37, pp. 13–15, Apr. 2004, IEEE.

[14] "Optical internetworking forum." `http://www.oiforum.com/`, accessed August 11, 2011.

[15] "EZchip technologies." `http://www.ezchip.com/`, accessed August 18, 2011.

[16] "PMC-Sierra." `http://www.pmc-sierra.com`, accessed August 18, 2011.

[17] "LSI corporation." `http://www.lsi.com`, accessed August 18, 2011.

[18] "Broadcom Corporation." http://www.broadcom.com, accessed August 18, 2011.

[19] "Xelerated." http://www.xelerated.com, accessed August 18, 2011.

[20] B. Wheeler and J. Bolaria, "Excerpt from: A Guide to Network Processors," tech. rep., Linley Group, April 2011. http://www.linleygroup.com/cms_builder/uploads/NPU_excerpt.pdf, accessed August 18, 2011.

[21] D. A. Patterson and J. L. Hennessy, *Computer organization and design: the hardware/software interface*. Amsterdam: Elsevier/ Morgan Kaufmann cop., 4th ed., 2009.

[22] "Birth of a Standard: The Intel 8086 Microprocessor." http://www.pcworld.com/article/146957/birth_of_a_standard_the_intel_8086_microprocessor.html, accessed September 13, 2011.

[23] "Intel corporation." http://www.intel.com, accessed August 3, 2011.

[24] Intel, "First the Tick, Now the Tock: Next Generation Intel Microarchitecture (Nehalem)," whitepaper, Intel, 2008. http://www.intel.com/content/www/us/en/architecture-and-technology/microarchitecture/intel-microarchitecture-white-paper.html; accessed August 2, 2011.

[25] Intel, "Intel Turbo Boost Technology in Intel Core Microarchitecture (Nehalem) Based Processors," whitepaper, November 2008. http://download.intel.com/design/processor/applnots/320354.pdf?iid=tech_tb+paper, accessed August 3, 2011.

[26] J. Charles, P. Jassi, N. Ananth, A. Sadat, and A. Fedorova, "Evaluation of the Intel Core i7 Turbo Boost feature," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pp. 188 –197, oct. 2009.

[27] Intel, "An Introduction to the Intel QuickPath Interconnect," whitepaper, Intel, January 2009. http://www.intel.com/content/www/us/en/architecture-and-technology/microarchitecture/intel-microarchitecture-white-paper.html; accessed August 2, 2011.

[28] D. Ziakas, A. Baum, R. A. Maddox, and R. J. Safranek, "Intel QuickPath Interconnect Architectural Features Supporting Scalable System Architectures," in *Proceedings of the 2010 18th IEEE Symposium on High Performance Interconnects*, HOTI '10, (Washington, DC, USA), pp. 1–6, IEEE Computer Society, 2010.

[29] "Sandy bridge microarchitecture." http://www.intel.com/content/www/us/en/architecture-and-technology/microarchitecture/intel-microarchitecture-codename-sandy-bridge.html, accessed August 3, 2011.

[30] O. Kahn, T. Piazza, and B. Valentine, "Technology Insight: Intel Next Generation Microarchitecture Codename Sandy Bridge," presentation, Intel, 2010. http://intelstudios.edgesuite.net/idf/2010/sf/ti/100913_SPCS001/index.htm, accessed March 6, 2012.

[31] G. Varghese, T. Piazza, and H. Jiang, "Technology Insight: Intel Next Generation Microarchitecture Codename Ivy Bridge," presentation, Intel, 2011. http://www.intel.com/idf/library/pdf/sf_2011/SF11_SPCS005_101F.pdf, accessed March 6, 2012.

[32] J. Brewer and J. Sekel, "PCI Express Technology," whitepaper, Dell Inc., 2004. http://i.dell.com/sites/content/business/solutions/whitepapers/en/Documents/wp-2004_pciexpress.pdf, accessed August 22, 2011.

[33] A. Bhatt, "Creating a PCI Express Interconnect," whitepaper, PCISIG, 2002. http://www.pcisig.com/specifications/pciexpress/technical_library/pciexpress_whitepaper.pdf, accessed August 22, 2011.

[34] R. Solomon, "PCI Express Basics," presentation, LSI Corporation, 2010. http://www.pcisig.com/developers/main/training_materials/get_document?doc_id=b4b7bc54bafa3e39535f9cdc0bd14d5d2eead8da, accessed August 22, 2011.

[35] HP, "Memory technology evolution: an overview of system memory technologies," whitepaper, Hewlett Packard, 2010. http://h20000.www2.hp.com/bc/docs/support/SupportManual/c00256987/c00256987.pdf, accessed September 14, 2011.

[36] D. Watts, A. Chabrol, P. Dundas, D. Frederickson, M. Kalmantas, M. Marroquin, R. Puri, J. R. Ruibal, and D. Zheng, *Tuning IBM System x servers for Performance*. IBM, 2009. http://www.redbooks.ibm.com/abstracts/sg245287.html, accessed September 22, 2011.

[37] E. Ciliendo, T. Kunimasa, and B. Braswell, "Linux performance and tuning guidelines," whitepaper, IBM, 2008. http://www.redbooks.ibm.com/abstracts/redp4285.html, accessed February 20, 2012.

[38] "X8dth-if user's manual," manual, Super Micro Computer Inc., 2010.

[39] R. Petersen, *Linux: The Complete Reference (6th Edition)*. Emeryville, CA, USA: McGraw-Hill Professional Publishing, November 2007.

[40] L. Torvalds and D. Diamond, *Just for fun : the story of an accidental revolutionary*. New York: HarperCollins, 2001.

[41] "The MINIX 3 Operating System." http://www.minix3.org/, accessed November 3, 2011.

[42] J. Corbet, G. Kroah-Hartman, and A. Rubini, *Linux device drivers*. Boston, MA: Safari Tech Books Online, 3rd ed., 2005.

[43] C. Benvenuti, *Understanding Linux network internals*. Sebastapol, California: O'Reilly, cop. 2006.

[44] R. Bolla and R. Bruschi, "Linux software router: Data plane optimization and performance evaluation," *Journal of Networks*, vol. 2, no. 3, 2007.

[45] "Ieee standard for information technology- telecommunications and information exchange between systems- local and metropolitan area networks- common specifications part 3: Media access control (mac) bridges," *ANSI/IEEE Std 802.1D, 1998 Edition*, pp. i –355, 1998.

[46] J. C. Mogul and K. K. Ramakrishnan, "Eliminating receive livelock in an interrupt-driven kernel," *ACM Trans. Comput. Syst.*, vol. 15, pp. 217–252, August 1997.

[47] J. H. Salim, R. Olsson, and A. Kuznetsov, "Beyond Softnet," in *Proceedings of the 5th Annual Linux Showcase & Conference*, 2001.

[48] Microsoft, "Receive-side Scaling Enhancements in Windows Server 2008," whitepaper, 2008. `http://msdn.microsoft.com/en-us/windows/hardware/gg463253`, accessed August 2, 2011.

[49] H. Krawczyk, "Lfsr-based hashing and authentication," in *Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '94, (London, UK, UK), pp. 129–139, Springer-Verlag, 1994.

[50] "Linux kernel documentation on IRQ affinity." `http://www.kernel.org/doc/Documentation/IRQ-affinity.txt`, accessed August 9, 2011.

[51] A. Foong, J. Fung, and D. Newell, "An in-depth analysis of the impact of processor affinity on network performance," in *Networks, 2004. (ICON 2004). Proceedings. 12th IEEE International Conference on*, vol. 1, pp. 244 – 250 vol.1, nov. 2004.

[52] "ixgbe driver readme file." `http://downloadmirror.intel.com/14687/eng/README.txt`, accessed September 27, 2011.

[53] "Netfilter." `http://netfilter.org/`, accessed March 9, 2012.

[54] S. Sidda, V. Pallipadi, and A. V. D. Ven, "Getting maximum mileage out of tickless," in *Linux Symposium*, (Ottawa, Canada), 2007.

[55] HP, Intel, Microsoft, Phoenix Technologies, and Toshiba, "Advanced configuration and power interface specification," specification, 2011. `http://www.acpi.info/DOWNLOADS/ACPIspec50.pdf`, accessed March 9, 2012.

[56] IBM, "Using the linux cpufreq subsystem for energy management," blueprint, IBM, 2009. `http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaai/cpufreq/liaai-cpufreq_pdf.pdf`, accessed March 9, 2012.

[57] J. Bonwick and S. Microsystems, "The slab allocator: An object-caching kernel memory allocator," in *In USENIX Summer*, pp. 87–98, 1994.

[58] "SLUB the unqueued slab allocator v6." `http://lwn.net/Articles/229096/`, accessed February 22, 2012.

[59] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel*. Boston, MA: Safari Tech Books Online, 3rd ed., 2005.

[60] S. Bradner and J. McQuaid, "Benchmarking Methodology for Network Interconnect Devices," RFC 2544, Internet Engineering Task Force, March 1999. http://www.ietf.org/rfc/rfc2544.txt; accessed July 20, 2011.

[61] S. Bradner, "Benchmarking Terminology for Network Interconnection Devices," RFC 1242, Internet Engineering Task Force, July 1991. http://www.ietf.org/rfc/rfc1242.txt, accessed July 28, 2011.

[62] "Supermicro X8STE motherboard." http://www.supermicro.com/products/motherboard/Xeon3000/X58/X8STE.cfm, accessed August 9, 2011.

[63] "Intel Xeon Processor x5650." http://ark.intel.com/products/47922/Intel-Xeon-Processor-X5650-(12M-Cache-2_66-GHz-6_40-GTs-Intel-QPI), accessed August 9, 2011.

[64] "Supermicro X8DTH-iF motherboard." http://www.supermicro.com/products/motherboard/QPI/5500/X8DTH-iF.cfm, accessed August 9, 2011.

[65] "Intel Ethernet X520 Server Adapters." http://www.intel.com/content/www/us/en/network-adapters/gigabit-network-adapters/ethernet-x520.html, accessed August 9, 2011.

[66] "ixgbe driver." http://sourceforge.net/projects/e1000/files/ixgbe%20stable/, accessed September 26, 2011.

[67] "Spirent TestCenter." http://www.spirent.com/Solutions-Directory/Spirent-TestCenter.aspx, accessed September 27, 2011.

[68] "Spirent TestCenter HyperMetrics CV module." http://www.spirent.com/Solutions-Directory/Spirent-TestCenter/HypermetricsCV.aspx, accessed September 27, 2011.

[69] "Ieee standard for information technology–telecommunications and information exchange between systems–local and metropolitan area networks–specific requirements part 3: Carrier sense multiple access with collision detection (csma/cd) access method and physical layer specifications - section one," *IEEE Std 802.3-2008 (Revision of IEEE Std 802.3-2005)*, pp. c1 –597, 26 2008.

[70] D. Rientjes, "Status of the linux slab allocators," presentation, Google, 2011. http://www.socallinuxexpo.org/scale9x/presentations/status-linux-slab-allocators.html, accessed March 26, 2012.

[71] L. Rizzo and M. Landi, "netmap: memory mapped access to network devices," in *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, (New York, NY, USA), pp. 422–423, ACM, 2011.

[72] "PF_RING." http://www.ntop.org/products/pf_ring/, accessed April 16, 2012.

A Single processor numerical results

A.1 Throughput in frames per second

Table 6: Single processor server throughput in frames per second.

Test case	**Throughput (frames per second)**						
Frame size (bytes)	**64**	**128**	**256**	**512**	**1024**	**1280**	**1518**
Baseline	2,934,272	2,997,602	3,082,230	3,049,352	2,934,272	2,653,927	2,083,333
Slab allocator	3,943,217	4,084,967	4,098,360	4,111,842	3,272,251	2,653,927	2,107,925
Netfilter disabled with slab allocator	7,183,908	7,164,531	7,022,472	6,009,615	3,134,770	2,490,936	2,083,333
Disabling power management features	2,934,272	2,997,602	2,955,082	2,983,293	3,033,980	3,033,980	2,997,602
Interrupt Throttle Rate 956	3,602,305	3,846,153	3,720,238	3,776,435	3,742,515	3,676,470	3,246,753
Receive ring length 64	3,602,305	3,846,153	3,720,238	3,776,435	3,676,470	3,623,188	3,246,753
Maximum performance	7,183,908	7,164,531	7,267,441	6,944,444	4,578,754	3,810,975	3,246,753

A.2 Throughput in Gbps

Table 7: Single processor server throughputs in Gbps.

Test case	**Throughput (Gbps)**						
Frame size (bytes)	**64**	**128**	**256**	**512**	**1024**	**1280**	**1518**
Baseline	1.97	3.55	6.81	12.98	24.51	27.6	25.63
Slab allocator	2.65	4.84	9.05	17.5	27.33	27.6	25.94
Netfilter disabled with slab allocator	4.83	8.48	15.51	25.58	26.18	25.91	25.63
Disabling power management features	1.97	3.55	6.52	12.7	25.34	31.55	36.88
Interrupt Throttle Rate 956	2.42	4.55	8.21	16.07	31.26	38.24	39.95
Receive ring length 64	2.42	4.55	8.21	16.07	30.71	37.68	39.95
Max perf	4.83	8.48	16.05	29.56	38.24	39.63	39.95

B Dual processor numerical results

B.1 Throughput in frames per second

Table 8: Dual processor server throughput in frames per second.

Test case	**Throughput (frames per second)**						
Frame size (bytes)	**64**	**128**	**256**	**512**	**1024**	**1280**	**1518**
Baseline	4,270,647	4,562,043	4,098,360	2,192,982	849,184	411,590	416,527
Slab allocator enabled	4,940,711	5,040,322	4,480,286	3,049,352	546,119	492,708	416,527
Netfilter disabled	4,940,711	4,791,491	3,591,954	1,137,397	478,927	600,961	325,097
Tickless disabled	4,270,647	4,562,043	4,480,286	4,496,402	4,340,277	2,949,663	3,132,832
Acpi processor setting disabled	4,612,546	4,325,259	4,480,286	4,432,624	4,416,961	3,787,878	3,246,753
Receive ring length 64	4,940,711	5,040,322	5,122,950	5,027,084	4,716,981	3,787,878	3,205,128
Max perf	10,080,645	10,245,901	10,162,601	8,004,543	4,545,454	3,731,343	3,156,565

B.2 Throughput in Gbps

Table 9: Dual processor server throughput in Gbps.

Test case	**Throughput (Gbps)**						
Frame size (bytes)	**64**	**128**	**256**	**512**	**1024**	**1280**	**1518**
Baseline	2.87	5.4	9.05	9.33	7.09	4.28	5.12
Slab allocator enabled	3.32	5.97	9.89	12.98	4.56	5.12	5.12
Netfilter disabled	3.32	5.67	7.93	4.84	4	6.25	4
Tickless disabled	2.87	5.4	9.89	19.14	36.25	30.68	38.55
Acpi processor setting disabled	3.1	5.12	9.89	18.87	36.89	39.39	39.95
Receive ring length 64	3.32	5.97	11.31	21.4	39.4	39.39	39.44
Max perf	6.77	12.13	22.44	34.07	37.96	38.81	38.84